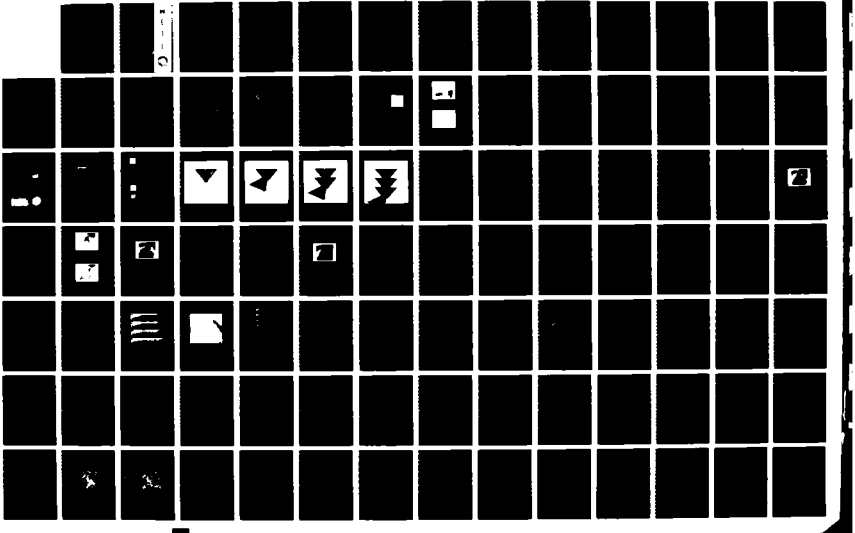
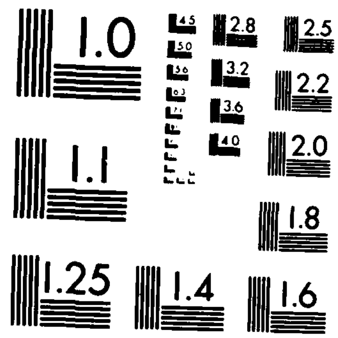
AD-A188 186 AN OVERVIEW OF VISION-BASED NAVIGATION FOR AUTONOMOUS LAND VEHICLES 1986.. (U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH S CHANDRAN ET AL. 1/2

UNCLASSIFIED APR 87 CAR-TR-285 ETL-0479 DACA76-84-C-0004 F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ETL-0479

An Overview of Vision-Based Navigation for Autonomous Land Vehicles 1986

Sharat Chandran
Larry S. Davis
Daniel Dementhon
Sven J. Dickenson
Suresh Gajulapalli
Shie-Rei Huang
Todd R. Kushner
Jacqueline LeMoigne
Sunil Puri
Tharakesh Siddalingaiah
Phillip Veatch

Center for Automation Research
University of Maryland
College Park, Maryland 20742

April 1987

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

E T L

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

A188 186

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: N/A
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CAR-TR-285 CS-TR-1831	5. MONITORING ORGANIZATION REPORT NUMBER(S): ETL-0479
6a. NAME OF PERFORMING ORGANIZATION: University of Maryland 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: U.S. Army Engineer Topographic Laboratories
6c. ADDRESS (City, State, and ZIP Code): Center for Automation Research College Park, MD 20742-3411	7b. ADDRESS (City, State, and ZIP Code): Fort Belvoir, VA 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable): ISTO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DACA76-84-C-0004
8c. ADDRESS (City, State, and ZIP Code): 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 623013E; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
An Overview of Vision-Based Navigation for Autonomous Land Vehicles 1986

12. PERSONAL AUTHOR(S) S. Chandran, L.S. Davis, D. Dementhon, S.J. Dickenson, S. Gajulapalli, S.R. Huang, T.R. Kushner, J. LeMoigne, S. Puri, T. Siddalingaiah, and P. Veatch

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO N/A	April 1987	117

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
17	07	
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Vision-Based Navigation
Parallel Algorithms

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes research performed during the first two years on the project Vision-Based Navigation for Autonomous Vehicles being conducted under DARPA support. The report contains discussion of four main topics:

1) Development of a vision system for autonomous navigation of roads and road networks.
2) Support of Martin Marietta Aerospace, Denver, the integrating contractor on DARPA's ALV program.
3) Experimentation with the vision system developed at Maryland on the Martin Marietta ALV.
4) Development and implementation of parallel algorithms for visual navigation on the parallel computers developed under the DARPA Strategic Computing Program--specifically, the WARP systolic array processor, the Butterfly, and the Connection Machine.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

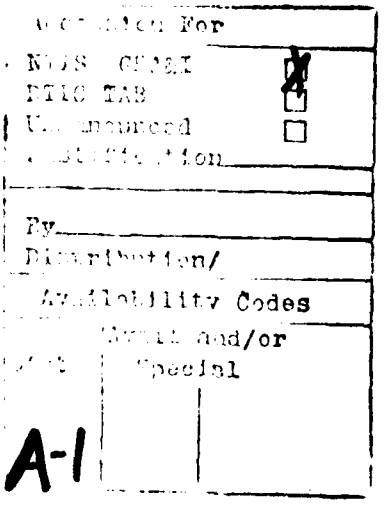
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Rosalene Holecheck	(202) 355-2767	ETL-RI-T

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

FIGURES AND TABLES



1. Introduction

This report describes research performed during the first two years on the project *Vision-Based Navigation for Autonomous Vehicles* being conducted under DARPA support. Our project, to date, has focused on four goals:

1) Development of a vision system for autonomous navigation of roads and road networks.
2) Support of Martin Marietta Aerospace, Denver, the integrating contractor on DARPA's ALV program.
3) Experimenting with the vision system developed at Maryland on the Martin Marietta ALV.
4) Development and implementation of parallel algorithms for visual navigation on the parallel computers developed under the DARPA Strategic Computing Program—specifically, the WARP systolic array processor, the Butterfly and the Connection Machine.

We have constructed, in our laboratory, a simulation facility for developing our visual navigation system. This facility consists of a robot arm carrying a sensor package over a terrain board. Roads and road networks are painted on the terrain board. The sensor package includes a black and white solid state camera, a structured light range scanner that provides range data registered with the black and white video data, and position encoders that allow us to control the height and attitude of the image sensors with respect to the terrain board. The simulation facility is described in greater detail in Section 2.1. While the imagery produced by the sensors on the robot arm is quantitatively different from data collected by the sensors on the ALV at Martin Marietta, we have found this simulator to be very valuable in designing the control components of the navigation system; these components account for the larger part of the actual code in

the system.

We have also developed a software simulation system for range data analysis. The simulator generates range images of a three dimensional polyhedral world. It is described in Section 2.2.

Section 3 of this report provides an overview of our visual navigation system. Figure 1.1 is a block diagram of the principal components of that system. The organization of the system was originally described in Waxman et al. [1]. The vision executive controls the activities of the other modules in the system. Its principal responsibilities are:

1) Controlling the focus of attention mechanism for image processing (i.e., identifying subsets of the sensed data likely to contain critical features for navigation and predicting both the photometric and geometric properties of these features).

2) Using knowledge stored in its visual knowledge base both to verify the model of its environment constructed on the basis of analyzing previous images and to extend that model further in the direction of travel.

The vision executive is described in more detail in Section 3.2. The image processing module (Section 3.3) contains algorithms for extracting structures from the image data that potentially correspond to objects of interest in the scene, such as boundaries of roads or obstacles on roads. The geometry module (Section 3.4) contains a variety of methods for reconstructing the three-dimensional geometry of the scene. The vision executive chooses the appropriate reconstruction technique either based on *a priori* knowledge about the environment or from

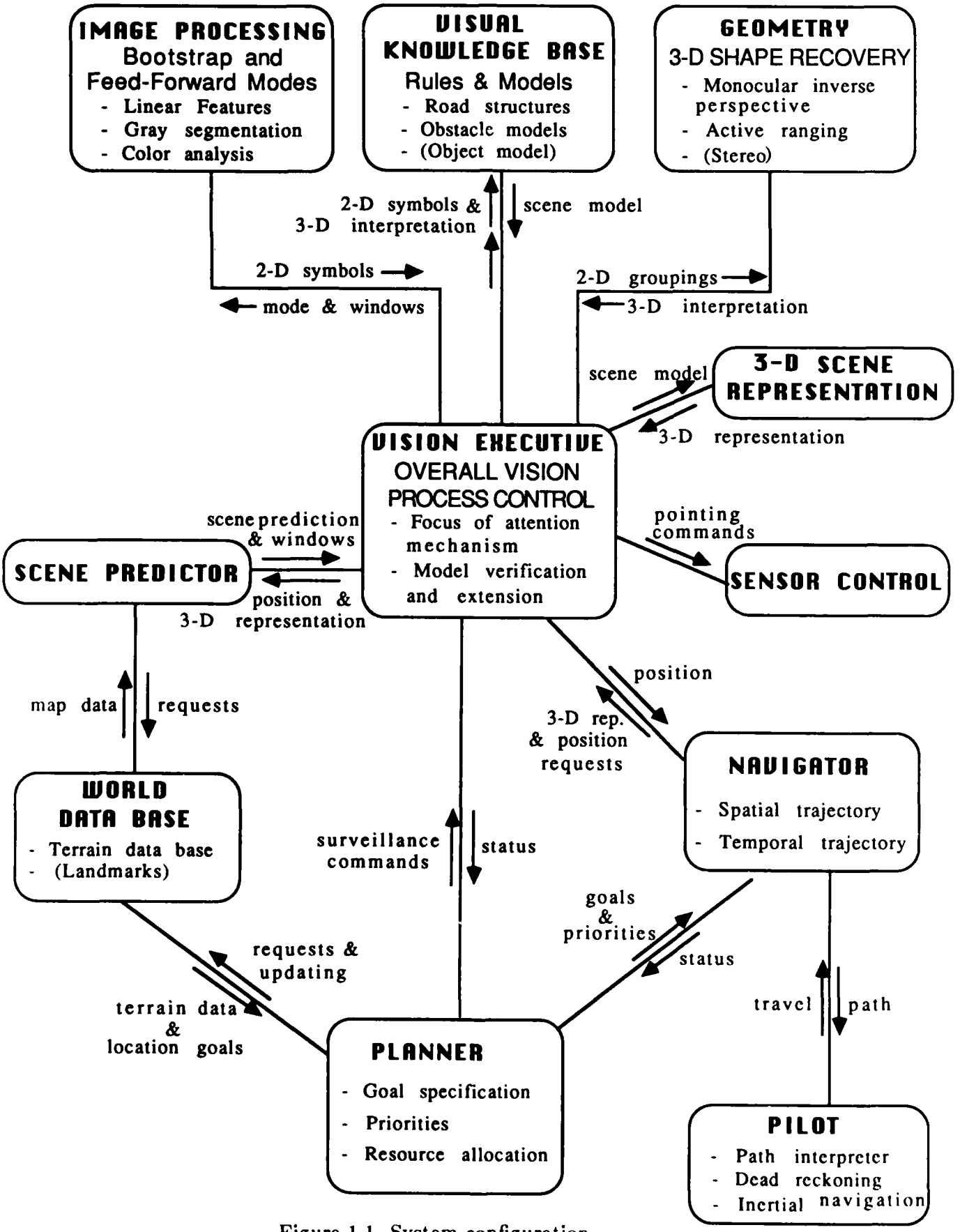


Figure 1.1 System configuration

its analysis of previously sensed data.

During the past several months we have extended the navigation system so that it can navigate over networks of roads. We have done this by incorporating a map database (Section 3.5) that includes a map of the roads on the terrain board. The organization of this database is identical to that of the road network map compiled by ETL for the Martin Marietta test site. The map is used by an extended scene prediction module (Section 3.6) to provide the vision executive with predictions about the appearance of the road(s) in its field of view. These predictions not only allow the vision executive to control its focus of attention mechanism, but also to control the viewing directions of its sensors to ensure that critical scene features are in their field of view (Section 3.7).

The planning module not only develops a prior route plan from the road map but also plans the local path of the vehicle through the three dimensional model constructed by the vision executive. Until now, this has been straightforward since our roads did not contain any obstacles. In more complex environments, of course, the planning problems become much more severe. We have developed a path planning system that utilizes a multiresolution representation of free space, obstacles and unknown space (shadows of obstacles and parts of the environment beyond the field of view of the sensors) that should be applicable to relatively complex planning problems. This path planning system is described in Section 4.

Our experience with the system described in Section 3 has made it clear that a more flexible framework is needed, principally for specifying the control aspects

of navigation. For example, our current system has a single strategy for tracking the boundaries of the road in an image—the scene prediction module identifies windows near the bottom of the image that will contain the projections of the left and right road boundaries, and after the image processing module identifies the actual locations of the projected road boundaries in those windows. Subsequent windows are placed that allow us to *continuously* track the road boundaries through the image. However, it would sometimes be advantageous to use some other strategy for tracking the road. For example, if prior knowledge indicated that the stretch of road currently being traversed were straight then one could imagine a strategy where after the initial windows were processed subsequent windows were placed relatively high up in the image, corresponding to segments of road relatively far from the vehicle. These and similar considerations have led us to design and develop an object-oriented system for visual navigation. A description of this system is included in Section 5.

One of the principal goals of the DARPA Strategic Computing Program is to understand how different models of parallel computation can be applied to problems in artificial intelligence. Towards this end, DARPA has sponsored both the hardware and software development of a variety of parallel machines, including the WARP[2], the Butterfly[3] and the Connection Machine[4]. The Computer Vision Laboratory currently has both a 16 processor Butterfly and a 128 processor Butterfly. A WARP is scheduled for delivery sometime during 1987.

In November of 1985, Todd Kushner spent several weeks at Martin Marietta, Denver, installing a version of our visual navigation system on the ALV

(these and other experiments are discussed in Section 7). Based on these experiments, we made certain modifications to the design of that system and then froze the design during the winter of 1986. Since then we have been working on a Butterfly implementation of most of the major components of that system. The Butterfly system performs the following operations in parallel:

1) window placement
2) Hough transforms for linear feature extraction (here we assume that the lowest level of visual processing is done on a machine such as the WARP and consider only the clustering aspects of linear feature extraction)
3) monocular inverse perspective
4) search for an acceptable scene model

Section 6 contains a description of the Hough transform Butterfly implementation, as well as some recent work on parallel search algorithms.

Finally, we have conducted a series of experiments both in the laboratory and on the Martin Marietta ALV. In Section 7.1 we describe recent experiments that involve navigating the robot arm through and into intersections, and in Section 7.2 we describe experiments performed on the ALV both in November 1985 and August 1986.

2. Simulation Systems

In this section we describe the laboratory facility that we have constructed to support the development of algorithms for visual navigation. Section 2.1 contains a description of the vehicle simulator, Section 2.2 describes the structured light range sensor, and Section 2.3 describes a range data simulation program.

2.1 Vehicle simulator

The vehicle simulator is a system which reproduces at reduced scale the interactions between the motion and the resulting changes in visual environment of a full-scale vehicle.

The four components of the system are:

1. A terrain model board, modeling the landscape that the robotic vehicle is likely to encounter;
2. A camera, positioned with respect to the terrain model in the same fashion as the vehicle camera would be positioned with respect to the world terrain;
3. A structured light ranger, designed to model the laser ranger of the vehicle and give images containing depth information;
4. A robot arm, which displaces the sensors across the terrain board in response to sensor data analysis software in the same fashion as the propulsion system of the full-scale vehicle would displace the vehicle sensors.

More details are now given about these four components of the simulator.

The robot arm is an American Robot arm, with six independent axes of movement, *waist, shoulder, elbow, wrist* (two axes), and *tool.* The motion of the tool plate has an accuracy of 1/1000 inch.

The arm is controlled by seven microprocessors; a Motorola 68000 processor supervises the six Motorola 6809 motion control processors. They interface with the user through the Magix operating system, a UNIX look-alike. The arm can be programmed with a specific robotic language, AR-BASIC, which also includes commands for communications with other computers such as the VAX computers of our laboratory.

The terrain model is constructed on a $4' \times 4'$ board. It models several paved road surfaces on various straight lines, s-curves, hills and intersection configurations. The camera-ranger system is displaced along these road segments by the robot arm, whose motions are computed by the programs run on the VAX computer on the basis of the image and range data that are gathered by the camera and the ranger and processed by the VICOM image processor.

The camera mounted on the robot arm is a Sony CCD array black-and-white camera. The camera images are input to a VICOM image processor controlled by a Motorola 68010 processor, or to a GRINNELL image processor controlled by a VAX 11/785 computer.

Mounted on the sides of the camera are three linear compression encoders which are used to maintain height, pitch, and roll of the camera in relation to the corresponding quantities of the simulated robotic vehicle. The signals from these encoders are processed by three of the seven analog-to-digital converters available with the robot arm hardware.

2.2 Structured light ranging system

The technology of laser rangers small enough to be mounted at the end of a robot arm is not available yet. Meanwhile a plane-of-light range scanner has been built to provide similar information and functionality. Its frame is mounted on the end plate of the robot arm, and supports the camera (Figure 2.1).

Figure 2.2 shows more details of the structure of the plane-of-light range scanner and its mounting with respect to the camera and the robot arm. The 150 W light generator is mounted on the robot arm itself and is not shown. The light is brought to the scanner by a four foot fiber optic light guide, which flares to a fine line of fibers. The line of light from the fiber optics line is further narrowed by a 4/1000 inch slit. The diverging light from this slit is refocused to a plane of light by a cylindrical lens. The plane of light is redirected out of the box of the device by a three-inch long mirror, whose angle is precisely controlled by a geared-down stepper motor. The stepping motor is connected to the VICOM computer through a controller card, and the command "STEP" from a VICOM Pascal program rotates the motor one step, giving a mirror angular displacement of 0.045 degrees. A full mirror revolution takes 8000 step commands.

The plane of light intersects an object in a planar world curve, and the image of this curve is seen from the camera. This image curve has a thickness of several pixels, and the median line of this curve is obtained. Each row of the image for a given position of the mirror is the image of a straight world line with a specific distance to the axis of the mirror, as shown in Figure 2.2. A two-

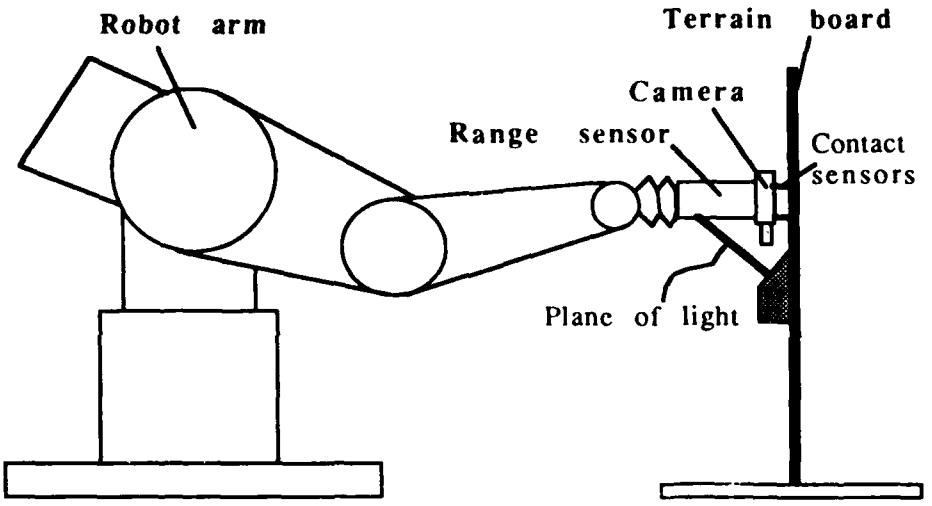


Figure 2.1 ALV simulator

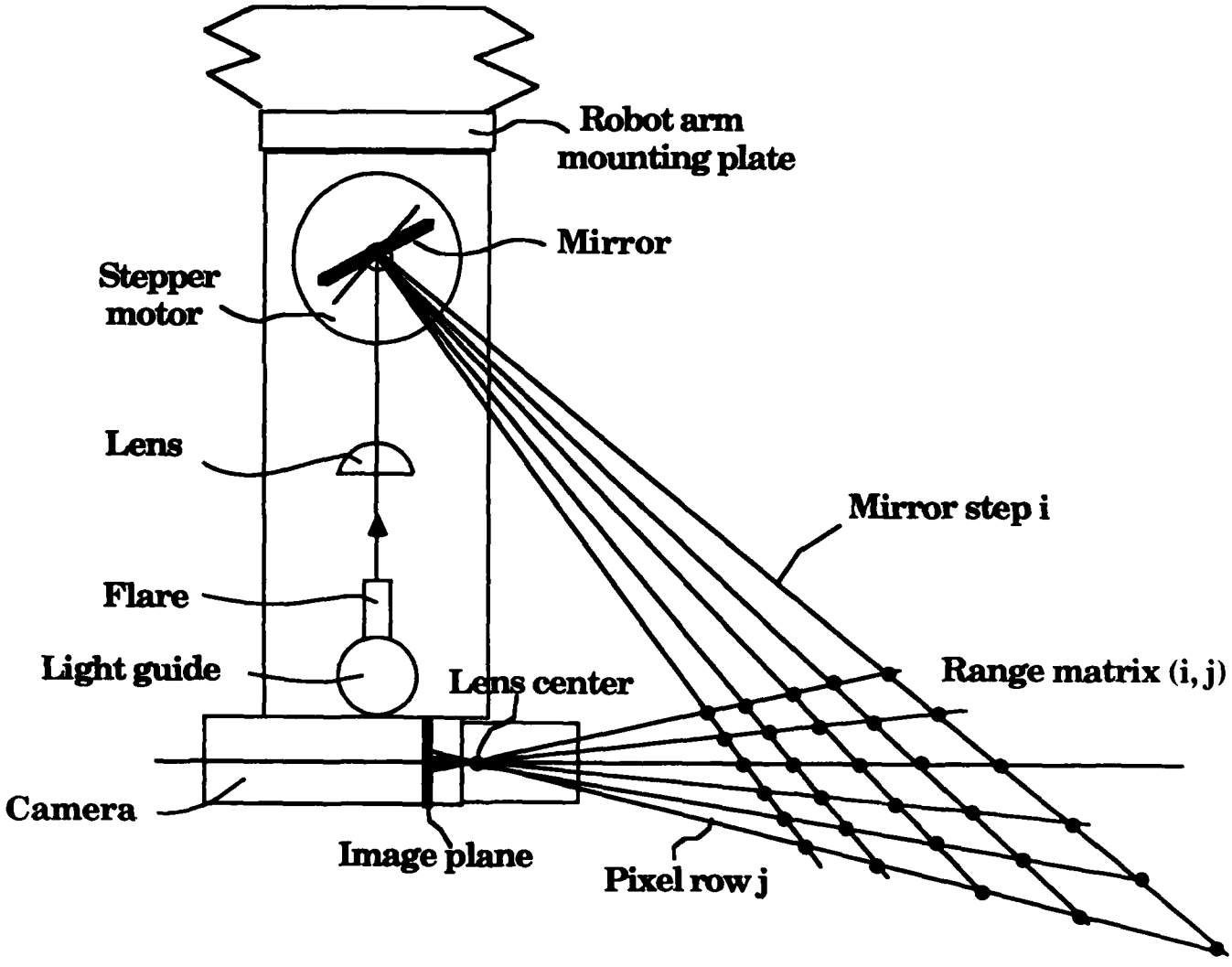


Figure 2.2 Plane–of–light range scanner mounted on robot arm

dimensional lookup table is calculated to give the ranges of all the world lines within the camera field of view and the mirror *field of range*. This lookup table maps the ranges of the intersection lines shown as dots in Figure 2.2.

The range acquisition algorithm is illustrated in Figure 2.3 and proceeds as follows:

1. While the mirror is at its home position facing the lens, a camera frame is grabbed. This is the initial image.

2. The mirror is moved from its home position to the initial scanning angle.

3. A camera frame is grabbed, which may contain the image of the intersection of the plane of light with a world object.

4. The difference between this image and the initial image is taken. Most of the information out of the image of the illuminated stripe is removed by this differencing operation.

5. The maximum brightness of the image of the stripe is calculated.

6. The image is thresholded to set the gray levels to zero out of the stripe (top right image of Figure 2.3).

7. Each image column is scanned until a non-zero pixel corresponding to a stripe boundary is found, then scanned further until a zero pixel corresponding to the other boundary is found. The row of the midpoint is calculated. The range of this pixel is found in the range lookup table at the appropriate mirror step column and pixel row line. The resulting range is placed in the range image being built, at the row indexed by the mirror steps, and column indexed by the image pixel column.

8. When all the image columns of a stripe image obtained for a given mirror step have been scanned, the row of the range image corresponding to this given mirror step is full, and the mirror is displaced to the next step. The program loops between items 3 to 8 until the stripe for the maximum step angle has been processed.

9. When the range image acquisition is completed, it is displayed, and the mirror completes its rotation up to its home position facing the lens.

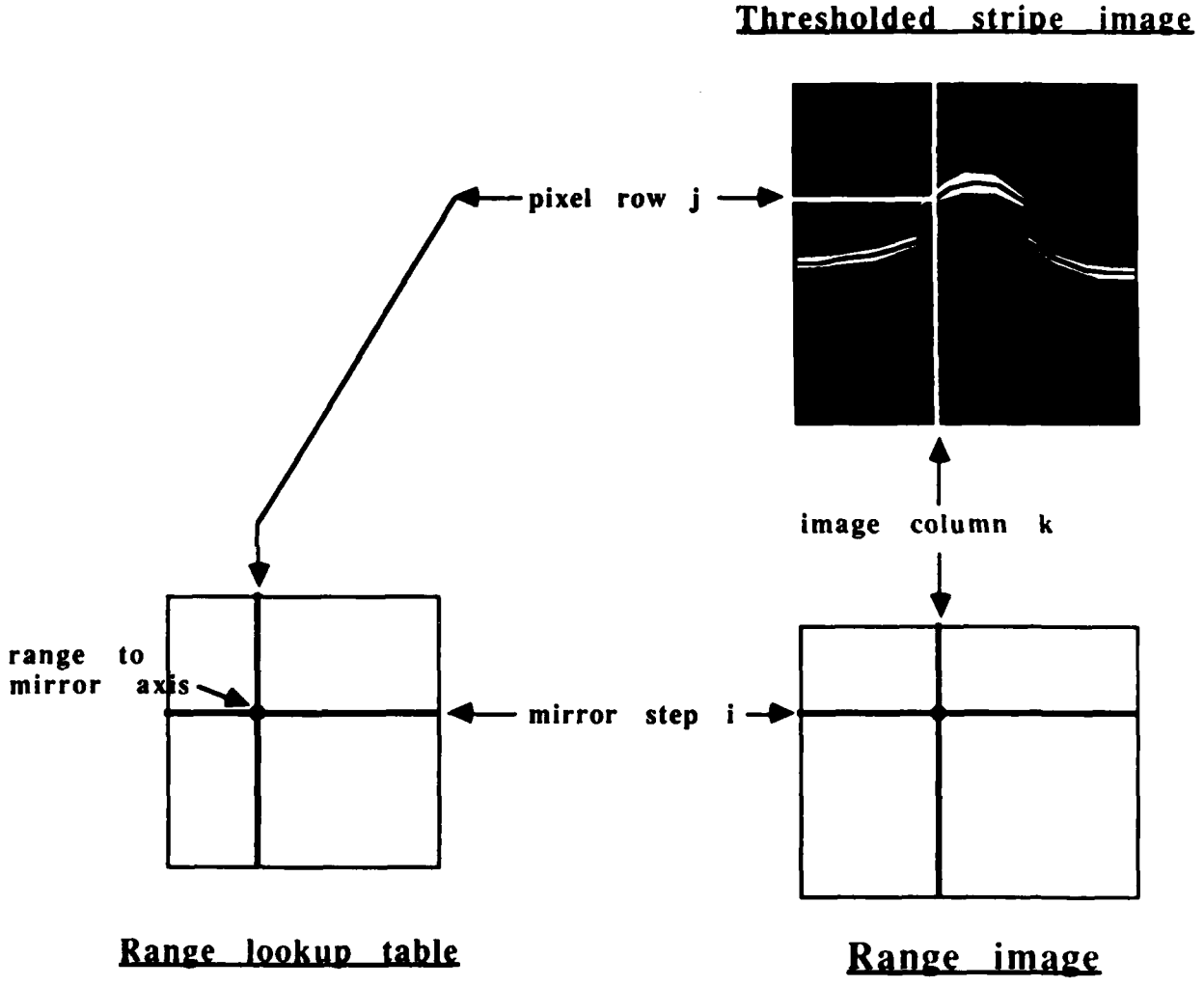


Figure 2.3 Creation of a range table from images of light stripes

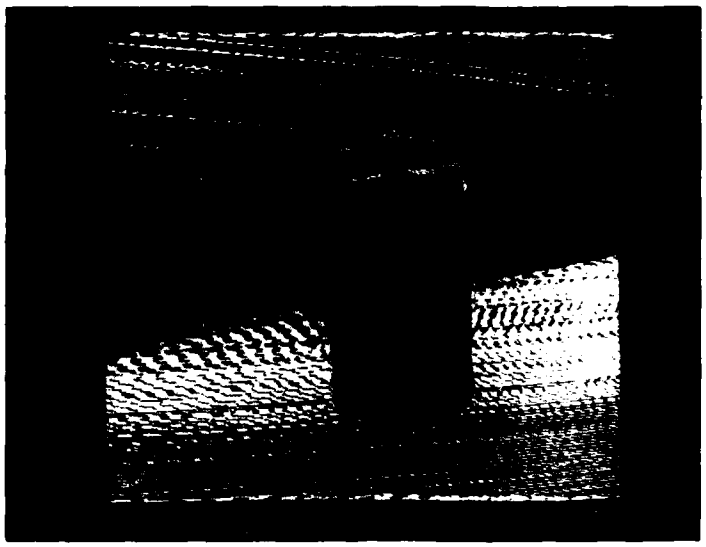


Figure 2.4a Range image in camera system

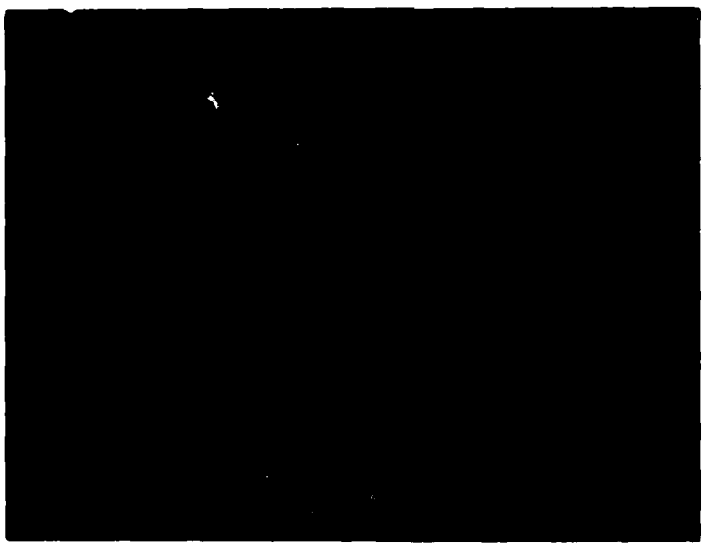


Figure 2.4b Range image in mirror system

Notice that this method for constructing the range is different from other structured light approaches. Previous work on structured light rangers constructed the range image in the camera coordinate system. Such a range image is shown in Figure 2.4.a. This image contains a spool of thread on a ground plane, with a vertical plane in the background. In such a range image, pixels between the images of the stripes are black because no range information could be obtained. It would be difficult to eliminate these gaps between the stripes without also smoothing the significant range discontinuities corresponding to the presence of object edges.

In the method proposed here, the problem of the gaps between stripes has been eliminated. The image construction method is based on the observation that if we position a camera with its lens center on the mirror axis, such a camera would see all the stripes as straight lines, even for stripes that the other camera sees as ellipse segments, because the images of the stripes with this new camera are contained in the plane of light which produces them. Straight stripe images obtained for different mirror steps are parallel and can be compressed to eliminate the gaps and build a continuous range image. Actually, a second camera is not required, since enough information from the original camera and mirror position is available to synthesize a range image as seen from the mirror's point of view. After stripe thinning, each straight stripe would be one pixel thick, so that each pixel row of the final image would correspond to a different stripe and therefore to a different mirror step. The range from a world point to the mirror can be obtained for each straight stripe pixel by the same triangulation procedure

which produced the range to the original camera. The range image acquisition algorithm presented above implements the proposed method and produces a range image from the mirror point of view.

Figure 2.4b is a range image obtained by this method for the same scene as Figure 2.4a. The spool looks more compressed because the aspect ratio of the picture depends on the mirror step angle, and also because the mirror was higher above the spool than the camera. Consequently the upper face of the spool and its holes are more visible.

A laser ranger would give a similar picture, because one picture row would also be obtained for each step of its vertical-scanning mirror. However, there would not be any shadow behind the spool. A shadow occurs in the structured light image because a part of the stripe of light was hidden from the camera by the spool, which prevented range calculation.

2.3 Range simulator

As an alternative to driving an ALV or directing a robot arm about a terrain board, we have developed an image synthesizer that allows us to create a world of objects and obtain range and visual images from any viewpoint in this world. The ability to make images from arbitrary positions is exactly equivalent to gathering images from an ALV that is driving through the synthetic world.

The synthesizer can model spheres, parallelepipeds, planar surfaces, cones, and cylinders. These objects can be translated and rotated in any fashion and may be positioned so that an object is partially or wholly inside of another object (an important property when constructing complex scenes from these basic building blocks).

After each object has been positioned (by multiplying it by a transformation matrix), a visual image is calculated based on a perspective projection in which the focal point is at the origin of the coordinate system and the image plane is placed in front of it at z = focal length.

Each pixel in the visual image has associated with it a grey level and a z-distance. The grey level can be created with the light source at any position. Surface reflections are assumed to be Lambertian and all objects have an equal albedo (it would be a simple extension to add variable albedos). No compensation is made for lowering intensity due to increased distance from the image plane and the light source.

From the visual image and the corresponding z distances we can calculate what the range image would appear to be. This initial range image has pixels that are spaced at equal intervals on the image plane. However, the ERIM range scanner produces images that are at equal angular intervals on the image plane so we convert to this format in order to accurately simulate the ALV's range scanning process. Interpolation of the initial range image is done using an intentionally crude algorithm to introduce noise into the system (of course, simply digitizing the image of the objects has introduced some noise, particularly for objects that have high curvature such as spheres). The final equiangular range image has all of the properties of an image produced by an ERIM scanner mounted on an ALV including the same field of view, eight-bit range values, and 64-foot ambiguity intervals.

The obstacle detection algorithms described in Section 3.3.2 are applied to the equiangular range image and the resultant binary obstacle image is mapped from spherical coordinates into the cartesian x-z ground plane. The ground plane map has initially four types of pixels: 1) obstacles or unnavigable terrain, 2) traversable terrain, 3) areas whose traversability is unknown because they are hidden by an obstacle (i.e.: shadow regions), and 4) areas whose traversability is unknown because they are outside of the field of view of the simulated laser sensor. The path planner will treat the ALV as if it were the size of a single pixel so a boundary the width of the ALV's radius is grown around all obstacle and shadow pixels.

At the start of the simulation the program requests the coordinates of the ultimate goal for the ALV. A straight line from the current location to this goal is plotted and a move along it is calculated. The endpoint of the move is passed to the path planner which tries to find a path through the ground map from the current location to the endpoint. The path planner uses a hierarchical algorithm based on a quadtree division of the ground map. The planner assumes that the vehicle can only travel through pixels that are marked as traversable.

If the planner fails to find a path to the first endpoint a series of heuristics are used in sequence to select alternate subgoal locations. Each subgoal is passed, one at a time, to the path planner until one is found that can be reached. If all of the heuristics are exhausted without a reachable subgoal being found, the program notifies the user and gracefully terminates.

Once the endpoint of the next move is found a transformation matrix is calculated that will place the origin of the coordinate system at this new location. This matrix, when applied to each object, will result in the next visual and range images being the scene that an ALV would see if it were driven to the endpoint. The matrix is fashioned so that the vehicle will be facing the ultimate goal location (other constraints on what direction the vehicle should be facing or how long each move should be are adjustable parameters in the program). If the move's endpoint is the same as the goal location the program terminates. Otherwise the transformation matrix is applied, the new visual image is found and the program begins another pass at moving the simulated vehicle toward its goal.

Figures 2.5 – 2.11 show some output from a typical simulation in which the modeled ALV drives approximately 200 feet to reach a goal location. There is no limitation on the distance from the starting point to the goal in the simulation. Figure 2.5 shows the visual images synthesized while traveling past several obstacles. A typical range image is shown in Figure 2.6. Figure 2.7 provides a key for Figures 2.8 – 2.11. These latter figures are the ground maps produced from the final range images at each step. Each map covers approximately 65000 square feet. The vehicle's current position is always at the center of the map.

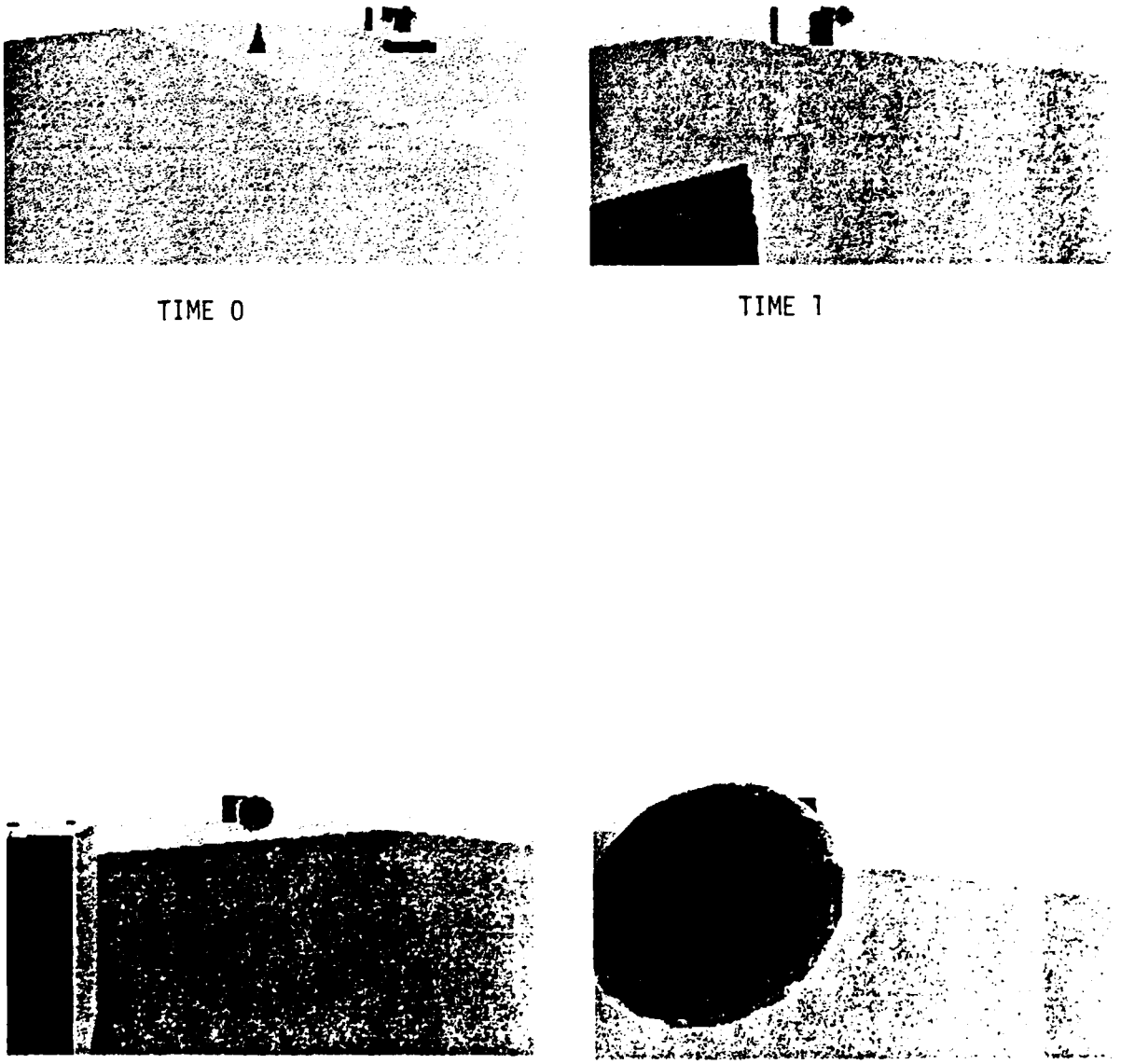


Figure 2.5 Simulated visual imagery

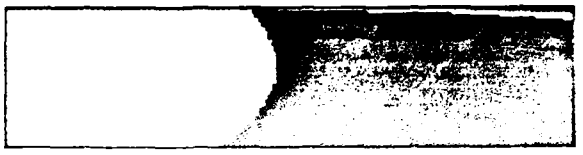
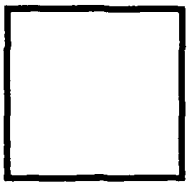
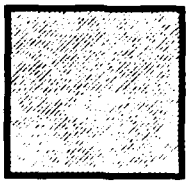


Figure 2.6 Typical range image (derived from time 3)

OUTSIDE OF THE SCANNER'S FIELD OF VIEW

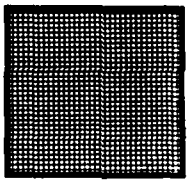


NAVIGABLE REGION



GROWN BOUNDARY REGION

OBSTACLE REGION



SHADOW REGION

Figure 2.7 Key for obstacle maps

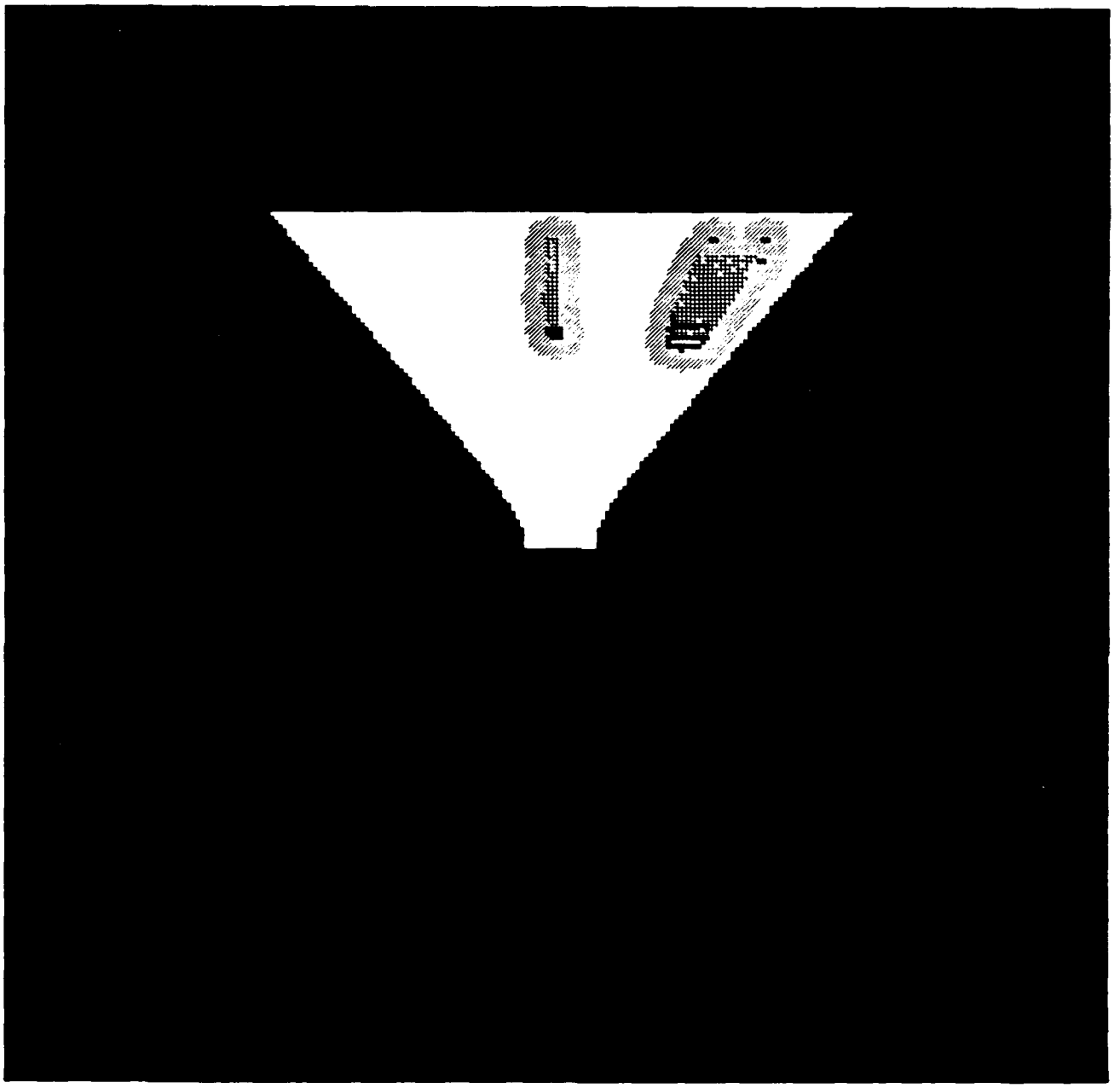


Figure 2.8 Obstacle map at time 0

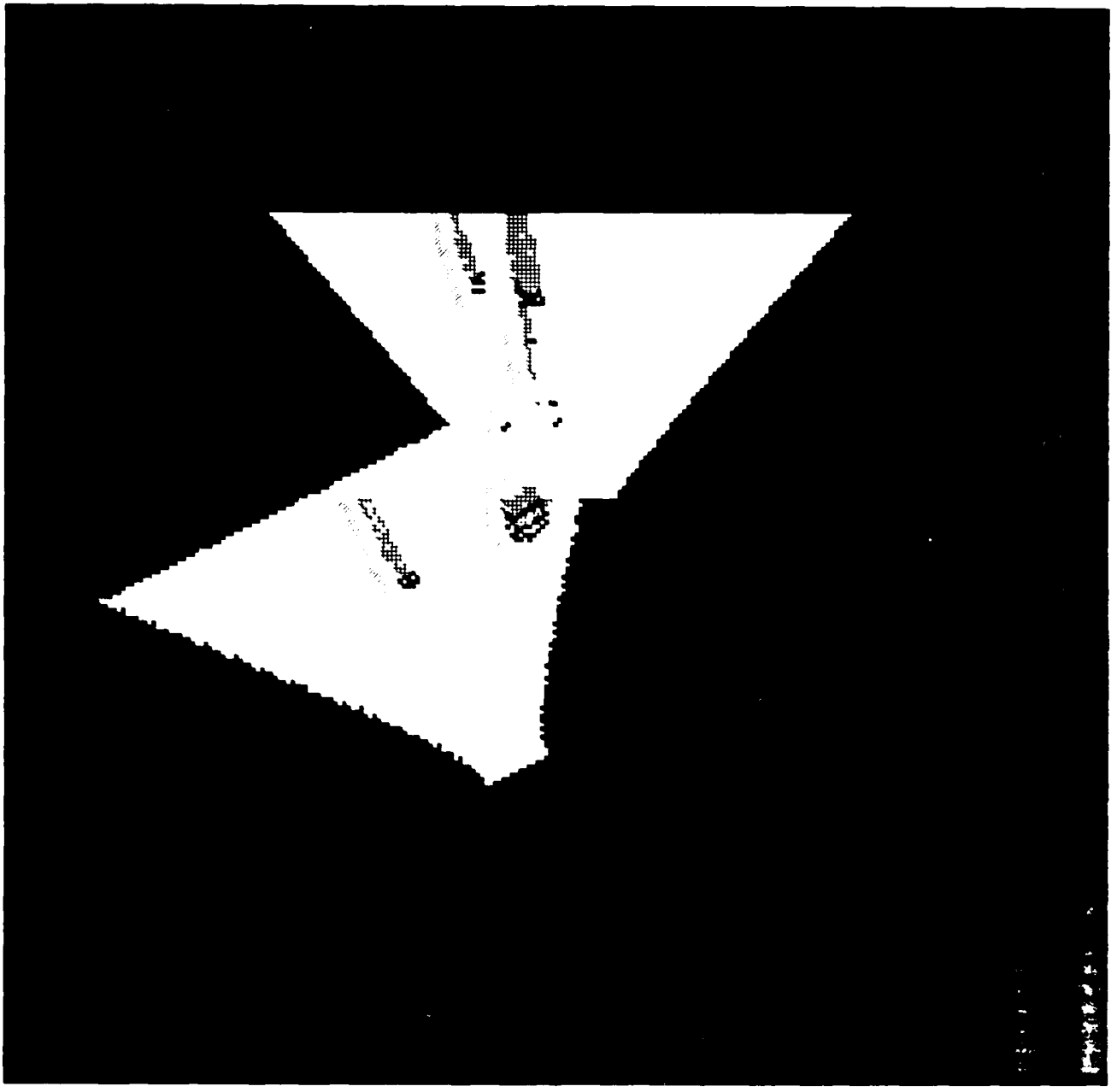


Figure 2.9 Obstacle map at time 1

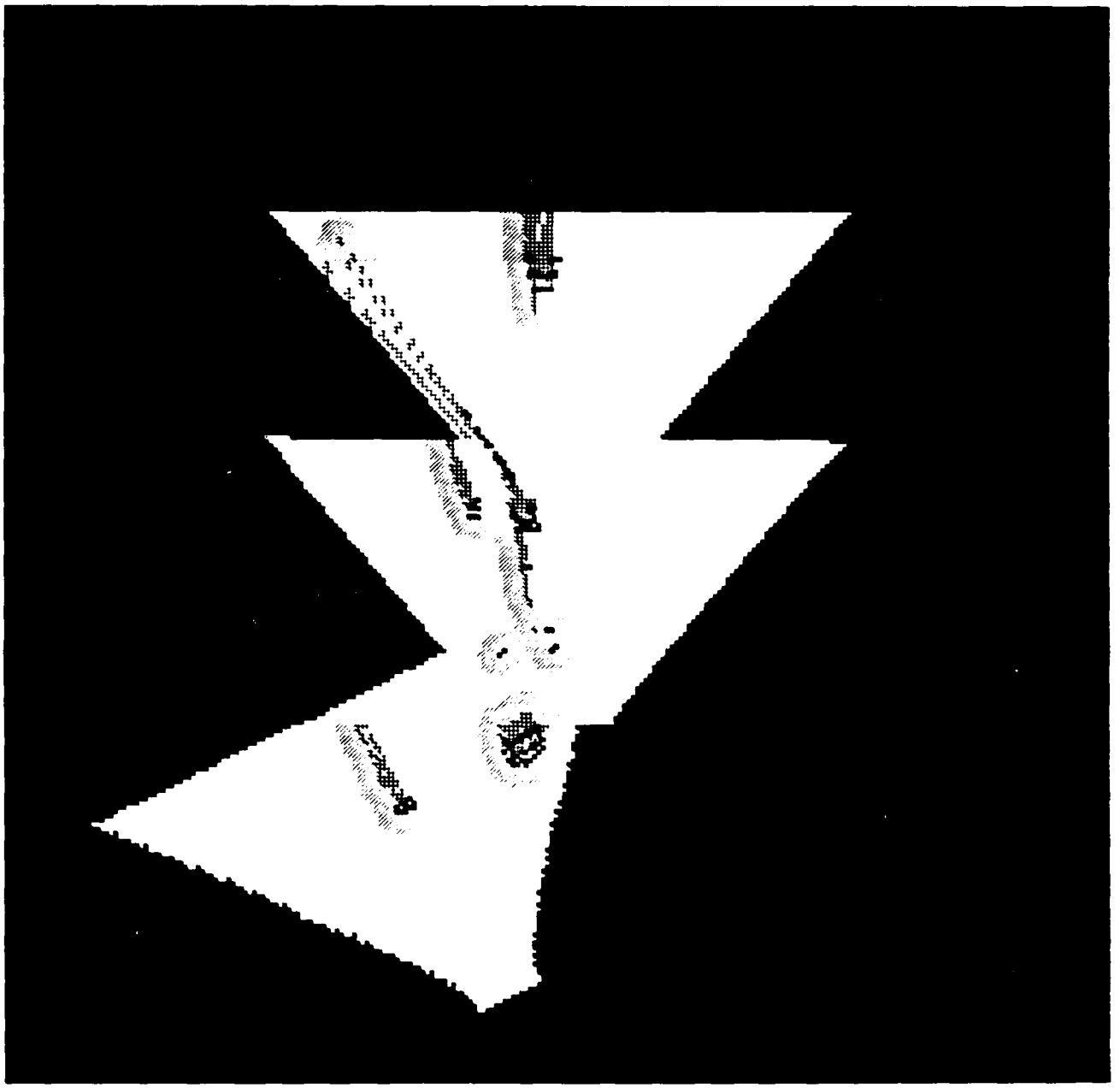


Figure 2.10 Obstacle map at time 2

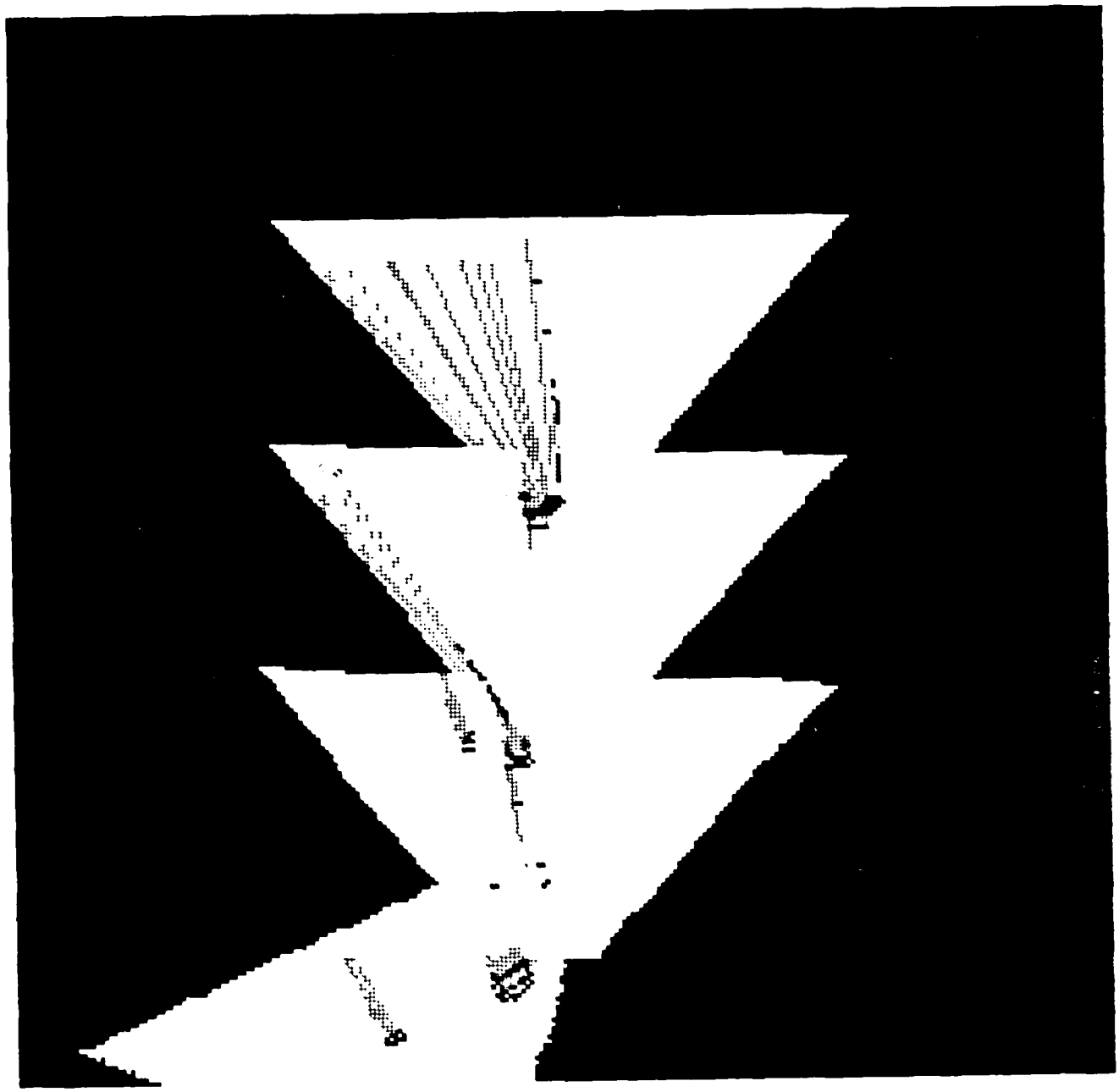


Figure 2.11 Obstacle map at time 3

3. Visual Navigation System

3.1 Introduction

This section contains an overview of the visual navigation system. This system runs on a VAX/VICOM environment, with image acquisition and some low level processing being done on the VICOM and the remainder of the vision processing, planning and navigation being performed on the VAX.

3.2 Vision executive

The principle responsibilities of the vision executive are:

1) controlling the focus of attention mechanism for sensor data processing, and
2) verifying and extending the sensor-based model of the environment.

Both of these tasks involve sensor control (for example, determining the appropriate pan and tilt for the video cameras), map database analysis and geometric reasoning about the three dimensional properties of scene objects identified in the sensor data.

Consider, for example, the verification of that portion of the scene model not yet traveled—i.e., still in front of the vehicle. Due to limitations of the vision and navigation systems this model will, to some degree, be inaccurate. Both to reduce these inaccuracies and to provide an anchor for analyzing previously unexplored parts of the scene, the vision executive analyzes those parts of the sensed data expected to contain components of the scene model. In order to do this, the vision executive must first decide, before acquiring an image, what parts of the

scene model it will search for. Then, based on the three dimensional properties of the scene model and a model for the assumed motion of the vehicle through that model, it can determine a viewing angle for the camera that will produce an image that will include those components. Currently this is done by trying to center the field of view of the camera on a point in the center of the road a fixed distance in front of the vehicle. The distance chosen is a function of both vehicle speed and the extent of the current three dimensional road model. One could imagine, of course, more sophisticated strategies for camera control based on other visual goals. The details of the sensor control module are contained in Section 3.7.

Once the direction of view of the camera is established, the vision executive can work with the scene predictor to establish the projected positions of key scene features in the next image frame. In the current implementation, the following types of predictions are supported:

1) Identify the points on the boundary of the image where the left and right road edges will enter the image.

2) Given a point x feet in front of the vehicle on the left (right) boundary of the road, will that point appear in the image and, if so, where?

The vision executive then places rectangular windows around these predictions and sends those windows to the appropriate sensor data processing module to extract the corresponding image object (straight edge segments of specific orientation and contrast sign for identifying road boundaries).

Currently, the "verification" stage of processing ends after the first two windows (one on the left boundary and one on the right boundary of the road) are processed since it is only for these two windows that expectations are generated concerning image properties such as orientation and contrast. Both to place and to analyze subsequent windows the vision executive applies either information derived from a road map, or, in the absence of a map, general knowledge about the geometry of roads, to generate subsequent predictions about the image locations of road features.

In the case where map data is available, the vision executive has available to it a set of precomputed predictions, indexed by world road coordinates, concerning the structure of the road in the immediate vicinity of the vehicle. Due to uncertainties in vehicle position and the limited accuracy of such map information the vision executive can only use this information in a qualitative fashion. For example, if the map database indicates that the road will be turning to the right, then the linear segments extracted by the image processing should turn towards the right in the image. Of more interest is the case where the map indicates that the vehicle is approaching an intersection. The images of intersections are quite complex, and the vision executive attempts, initially, to avoid processing the specific part of the image where the intersecting roads actually meet until it has identified parts of the image containing the constituent roads. It does this by identifying windows predicted to include the intersection, and then searching around these windows for the intersecting roads. The information potentially available to the vision executive concerning the structure of the intersection

includes the spectral properties of the roads, their widths and the angles between the roads at the intersections. A detailed example of navigating through an intersection on the terrain board is included in Section 7.

3.3 Image processing

3.3.1 Video image processing

The image processing module transforms an input image into a symbolic representation of the boundaries of the roads in the field of view by extracting dominant linear features from grey-level or color imagery. Different representations can be used in the image domain: boundary-based and region-based are two examples of such representations. We present two kinds of algorithms for extracting roads from imagery, corresponding to these two different representations: linear feature extraction and grey-level or color segmentation.

3.3.1.1 Thresholding algorithms

In this section we present algorithms relying primarily on segmenting grey-scale and color images to locate dominant linear features in the feed-forward mode of operation. The first uses a prediction of where the road boundaries will appear in small windows at the bottom of the image, along with a boundary-based representation of the linear features in those windows, to collect statistics of grey scale or color values on the surface of the road.

Initial windows covering segments of the left and right boundaries of the road are chosen based on projecting the current 3-D road model onto the image plane and determining where the road boundaries enter the image. For these windows we estimate the orientation and position of the projection of the road edges using a Hough transform as described in [5]. After the projection of the

road edges are determined in each window, the two regions of each window separated by those edges, presumably road and non-road, are histogrammed separately. A minimum-error threshold for each window is then computed from these histograms. The thresholds are then each applied to half of the image to segment the image into *road points* and *non-road points*.

Processing differs for subsequent windows by constraining the lines in these windows to connect to the lines in the immediately preceding windows (the *road continuity* assumption)—e.g., an endpoint of the line in the previous window becomes the *pivot,* or intercept, of the line in the current window. Furthermore, we constrain the orientation of the line in the current window to be in a small interval, $[\theta_m, \theta_M]$, centered about the orientation of the line in the previous window. Since the pivot point is fixed, we need only estimate one parameter—the direction, θ, of the line through the pivot point. Given the pivot point, (x_p, y_p) and any road point (x_v, y_v) in the window, the angle θ of the road point with respect to the pivot is simply

$$\theta = \tan^{-1}((y_v - y_p)/(x_v - x_p)).$$

If the values of θ for road points in a window are accumulated, we would expect the counts to suddenly drop off at a value of θ roughly corresponding to the line segment best fitting the road boundary. The angle corresponding to the accumulator value closest to a fixed threshold is chosen as the boundary line. To prevent the line from overshooting beyond the actual road boundary, the line is cut back to the road point furthest along the line from the pivot. Figure 3.1 shows the algorithm applied to an image in feed-forward mode.

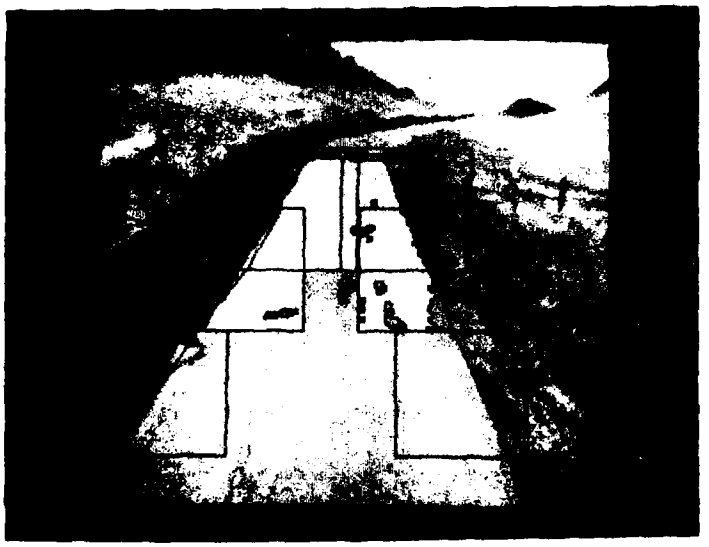


Figure 3.1 Thresholding algorithm applied to an image from the Martin Marietta test site

Since the determination of the orientation and position of the projection of the road edges in the initial and subsequent windows using the Hough transform is expensive, a second algorithm was developed that relies only on the initial windows containing portions of road and non-road around the projection of the current 3-D road model onto the image. A threshold is calculated for each window by sampling two populations of the window assumed to contain only road and non-road pixels, in two small diagonally opposite corners of the window, and choosing a threshold satisfying a minimum-error criterion for the two samples. The left and right boundaries of the road in the image are extracted using the thresholds. Line segments are fit to the border by determining for each window the road/non-road border point along the top, middle, and bottom window rows that satisfies the following minimum-error criterion: the total of the fraction of road points on its non-road side of the window row and the fraction of non-road points on its road side of the window row is at a minimum. Figure 3.2a shows the result of applying this algorithm to an image from the Martin Marietta test site.

Contrast reversal can occur across the road boundary, causing simple thresholding segmentation to fail. A third algorithm calculates thresholds by sampling a population of the window assumed to only contain road pixels, in the lowest corner closest to the center of the road, and selecting a range of threshold values centered about the mean of the sample of size equal to a constant number of standard deviations of the sample. Figures 3.2b and 3.2c show the second and third algorithms applied to an image that exhibits contrast reversal.

Figure 3.2a The second thresholding algorithm applied to an image from the Martin Marietta test site

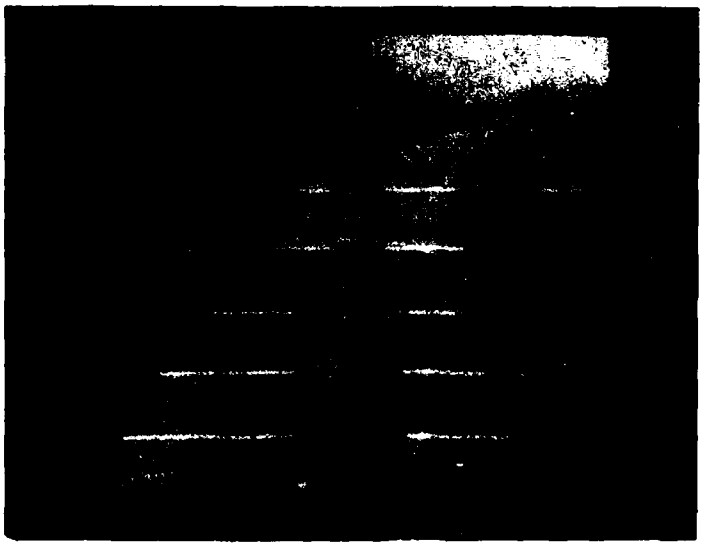


Figure 3.2b Same algorithm as 3.2a applied to an image with contrast reversal

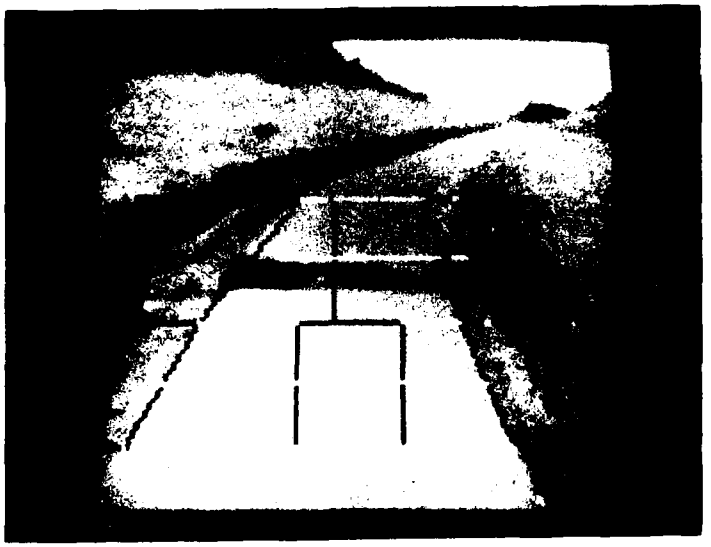


Figure 3.2c A third algorithm applied to an image with contrast reversal

3.3.1.2 Multiresolution algorithms

To improve region-based segmentation of grey scale and color images, the image can be blurred to reduce the effects of background texture and minor variations on the surface of the road. A better approach, however, is to reduce the image size if possible. In our experiments, good segmentations were obtained for road images with 64 by 64 pixels. Another improvement over the above methods that select left and right boundary road edges separately is to select the left and right road edges simultaneously constrained by knowledge about the geometric properties of the road, e.g., road width.

This algorithm for locating dominant linear features in an image at different resolutions in feed-forward mode of operation uses knowledge about the predicted location of the road to position a window at the lower middle part of the road assumed to contain only road pixels. Statistics of grey scale or color in this window are computed and a range of threshold values are selected about the mean of the sample of size equal to a constant number of standard deviations of the sample. The entire image is segmented using these threshold values. An initial window covering segments of both the left and right boundaries of the road is chosen based on projecting the current 3-D road model onto the image plane and determining where the road boundaries enter the image. Line segments are fit to the borders by simultaneously determining for the window the road/non-road border points along the top, middle, and bottom window rows that satisfy two minimum-error criteria: the total of the fraction of road points outside the border points and the fraction of non-road points between the border points is a

minimum, and the distance between the border points is closest to the predicted distance between the left and right boundaries of the road projected from the current 3-D road model onto the image at that line.

Once a window is processed, the next window is chosen in such a way that the length of both line segments found in this extrapolated window is maximized. This is currently done by placing the next window so that its center lies on the road centerline projected from the previous window. Processing is automatically stopped when the window is within 20% of the top of the image, when the window leaves the image, or when no left or right window border points are found. Figure 3.3 shows the results of applying this algorithm to a road image from the Martin Marietta test site.

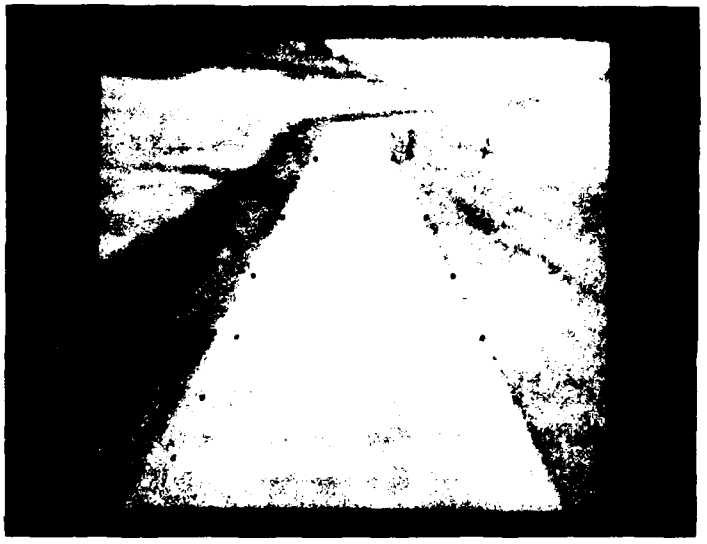


Figure 3.3 Multiresolution algorithm applied to an image from the Martin Marietta test site

3.3.2 Range data processing

A surface may be considered navigable if its slope is sufficiently small. In the extreme case of an obstacle sticking up from the ground the area will contain a discontinuous slope that, when digitized, will appear as a very steep slope and hence mark the obstacle as being an unnavigable area. The slope is measured by calculating the first derivative of the range with respect to the vertical scanning angle and the first derivative of the range with respect to the horizontal scanning angle. These calculations involve nothing more than addition and subtraction and so can be performed very quickly. Obstacle detection from first derivatives has the desirable property of being less sensitive to uncertainties in the range scanner's orientation than algorithms based on the range itself.

The range scanner is mounted on the ALV approximately nine feet above the ground and looks out in the direction that the vehicle is traveling. An ERIM range image is most naturally described in a spherical coordinate system (see Figure 3.4) in which the 64 rows of the image are at equally spaced values of the vertical scan angle, ϕ, and the 256 columns are at evenly spaced values of the horizontal scan angle, θ. The image has a 30 degree vertical field of view ranging from approximately 6 degrees beneath the horizon to 36 degrees beneath the horizon. The 80 degree horizontal field of view extends from about 130 degrees to 50 degrees as measured from the x-axis shown in Figure 3.4.

The range at each pixel in a range image is calculated in hardware from the wave phase difference of a modulated laser beam. This causes the calculated ranges to be the true range modulo 64 feet, i.e.: a value of 10 feet may indicate

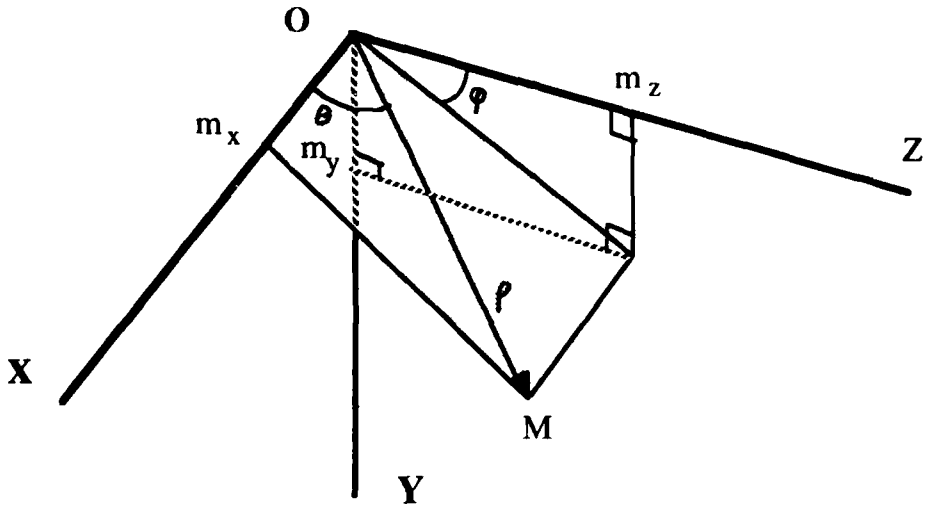


Figure 3.4 Range scanner geometry

that the laser is returning from 10 feet or 74 feet or 138 feet, etc. The resultant ambiguity must be compensated for by any obstacle detection algorithm.

Due to the ambiguity effect, output range values are all between zero and 64 feet. They are quantized into three inch units so that the final output of the scanner is a 64×256 array of 8 bit values ranging from zero to 255.

The geometry of the range scanner results in the following relationships:

$$x = \rho \cos\theta \tag{1}$$

$$y = \rho \sin\theta \sin\phi \tag{2}$$

$$z = \rho \sin\theta \cos\phi \tag{3}$$

If θ is held constant then the slope in the z direction is,

$$\frac{\Delta y}{\Delta z} = \frac{\frac{\Delta\rho}{\rho}\frac{\tan\phi}{\Delta\phi} + 1}{\frac{\Delta\rho}{\rho}\frac{1}{\Delta\phi} - \tan\phi} \tag{4}$$

If ϕ is held constant then the slope in the x direction is,

$$\frac{\Delta y}{\Delta x} = \frac{\frac{\Delta\rho}{\rho}\frac{\tan\theta}{\Delta\theta}\sin\phi - \sin\phi}{\frac{\Delta\rho}{\rho}\frac{1}{\Delta\theta} - \tan\theta} \tag{5}$$

Excluding the terms in equations (4) and (5) that we know a priori, we see that the changes in height in the x and z directions are a function of $\Delta\rho/\rho$. If we used some approximation of ρ we would have a direct relationship between the easily calculated $\Delta\rho$ for a fixed θ or ϕ at a pixel and the slopes at that pixel. Our experiments with real range data suggest that the following is an adequate approximation:

$$\rho \approx \frac{H}{\sin\theta \sin\phi} \tag{6}$$

where H is the height of the range scanner above the ground. Equation (6) comes from substituting H for y in equation (2). In hilly terrain this approximation is probably not adequate but it works well for many scenes and the derivative algorithm that uses this approximation is less sensitive to orientation errors than other algorithms of similar simplicity and speed.

Using equation (6) we can calculate what $\Delta\rho$ would be at each pixel if the slopes were zero. The difference between this predicted $\Delta\rho$ and the actual $\Delta\rho$ found in a range image is a measure of the actual slope. Large differences between predicted and actual $\Delta\rho$'s will be formed by edges of objects as well as surfaces with steep slopes. Thresholding the absolute values of these differences yields pixels that are likely to be obstacles.

We estimate the $\Delta\rho$'s at the (i,j)'th pixel in the image by averaging over a 3×3 neighborhood:

for *the θ derivative*,

$$\Delta\rho = \sum_{k=i-1}^{i+1} Range[k,j+1] - \sum_{k=i-1}^{i+1} Range[k,j-1] \tag{7}$$

for *the ϕ derivative*,

$$\Delta\rho = \sum_{k=j-1}^{j+1} Range[i+1,k] - \sum_{k=j-1}^{j+1} Range[i-1,k] \tag{8}$$

The term "θ derivative" is used loosely to mean $\Delta\rho$ calculated for a known $\Delta\theta$ while ϕ is held constant. Similarly, "ϕ derivative" means $\Delta\rho$ calculated for a

known $\Delta\phi$ while θ is constant.

One could of course simply threshold the actual $\Delta\rho$'s without first subtracting the expected $\Delta\rho$'s and assume that large $\Delta\rho$'s indicate surfaces that have steep slopes and hence are not navigable. This approach however would severely reduce one's capability to detect obstacles. A perfectly flat surface will yield a $\Delta\rho$ for the ϕ derivative of about 30 if it is 60 feet away but the same surface at a range of 10 feet only has a $\Delta\rho$ of about 0.9 (these values are the unnormalized sums of the 3×3 neighborhood calculation shown in equation (8)). This wide range in $\Delta\rho$'s leaves any thresholding algorithm in a bind. Small threshold levels would find nearby obstacles but farther away flat surfaces would be falsely labelled as obstacles. Conversely, larger threshold levels would hide significant obstacles that were near the range scanner. What is needed is a variable threshold setting. This approach points out another way of looking at the range derivative algorithm: we are, in essence, creating a variable threshold that changes across an image based on expected $\Delta\rho$'s. While this simplistic view is a useful description, the derivative algorithm is founded on the mathematical relationships between $\Delta\rho$ and a surface's slopes and is not a randomly chosen heuristic for setting variable threshold levels.

Once an obstacle is detected one wishes of course to know where it is. This raises another interesting problem arising from the manner in which the range was found by the scanner. As the laser beam goes out from the scanner it diverges in the shape of a cone. The return signal that determines how far the beam traveled is a sum of all of the ranges within the cone. For example, if half

of the cone hits a tree and the other half traveled on to the ground the returning signal would yield a value that is somewhere between the distance to the tree and the longer distance to the ground. This is called the mixed pixel problem.

To counter this effect we try to avoid mixed pixels that occur at the edge of obstacles in estimating object location and use the more accurate range values in an object's interior area. The location of the interior is determined by the sign of the derivative. If an obstacle is sticking up from the ground then a negative θ derivative indicates that the pixel is on the left edge of an obstacle so the range is taken from the pixel that is to the right of the current pixel. The same approach can be used with the ϕ derivative for determining if one is at the top (negative derivative) or bottom (positive derivative) edge. We have found that this strategy works well for avoiding false ranges due to mixed pixels.

The ambiguity interval problem has been approached in two different ways. For actual range images taken from the ALV we have found that if the vehicle is not traveling over very hilly territory, simply deleting the upper few rows of the image removes most of the pixels that are beyond the first ambiguity interval ($\geq$ 64 feet). This does not affect the obstacle detection algorithm significantly because these pixels tend to be very mixed anyway (the beam has spread out into a relatively wide cone by the time it has traveled beyond 50-60 feet) and so are of little use for accurate obstacle detection.

A more time consuming but somewhat more precise approach has been to examine each column from bottom to top in the image. Whenever adjacent pixels go from large values suddenly to very small values it is reasonable to assume

that an ambiguity interval has been reached and that all pixels in the column beyond this point should have 256 added to their ranges. This approach was used by the range image synthesizer described in Section 2.

Uncertainties in the ALV's orientation with respect to the ground introduces errors into calculations of ranges and range derivatives. We have analyzed these errors and shown that the derivative approach is significantly less sensitive to these uncertainties than the range difference method which simply computes the (absolute) difference between observed and predicted ranges. Our derivative algorithm is also less sensitive than the height difference method that first converts ranges to heights (i.e. y coordinates) and then subtracts the expected constant height. The three algorithms were compared by applying them to the range image of a flat plane in which the image was taken by a scanner rotated in some manner. Table 3.1 summarizes the results for four types of rotational perturbations: 1) a three degree error in the vertical angle ϕ, 2) a three degree error in the horizontal angle θ, 3) a three degree error in rotation about the z axis (roll error), and 4) the combined effect of three degree errors in ϕ, θ, and roll. The table contains two entries for each combination of algorithm and perturbation. The top entry is the largest absolute value in the entire image and represents a worst-case scenario. In many scenes, however, the road will be near the center of the image's horizontal field of view and large errors on the periphery are not critical. This scenario is represented by the bottom entry which is the largest absolute value within the central 30 degrees of the image (i.e. $85 \leq \theta \leq 105$). The derivative algorithm is broken down into its ϕ derivative and θ derivative steps in the

table.

Several important trends emerge from Table 3.1. The θ derivatives were very insensitive to all four rotational perturbations. The ϕ derivatives were somewhat more sensitive. When the entire image was considered, the maximum ϕ derivative errors for each rotation were always at least 25% less than the maximum height difference errors. Within the central 30 degrees of the horizontal field of view, the maximum ϕ derivative errors were 45% - 75% less than the maximum height difference errors. The range difference algorithm was very sensitive to all forms of rotations. In several instances the range difference errors were a full order of magnitude larger than the derivative errors. These results clearly show that the derivative algorithm is more robust under rotational uncertainties *than either the height difference or the range difference* algorithms.

Figure 3.5a shows a montage of four range images. The images were produced by an ERIM range scanner while mounted on the ALV at Martin Marietta's Denver test track site. Moving down from the top of the figure, each image is taken five feet further down the road. The person in the last image moved to his position after the first three images were taken and so does not appear in the first three images. Figure 3.5b is a visual picture taken by the ALV at the same time as the top range image in Figure 3.5a. The box in the lower right corner of Figure 3.5b can be seen in the top two images of Figure 3.5a. The cone that is on the right side of the road about half way up Figure 3.5b is present in all four range images. The cone is not very apparent in Figure 3.5a but it is clearly marked as an obstacle in Figure 3.6a, which is the

thresholded output of the θ derivative algorithm. Figure 3.6b is the thresholded output of the ϕ derivatives. The small, dark blob that is located in the center columns of the top few rows of each image in Figure 3.5a and that is marked as an obstacle in Figures 3.6a and 3.6b does not correspond to any actual object in the scenes but rather is spurious data generated by the range scanner's electronics.

Table 3.1: Comparison of Obstacle Detection Algorithms for Sensitivity to Scanner Perturbations

Perturbation:	Magnitude of Errors: (1)			
	Theta Derivative Algorithm	Phi Derivative Algorithm	Height Difference Algorithm	Range Difference Algorithm
3 degrees in horizontal angle	0.8 (2) 0.3 (3)	1.5 0.4	5.1 1.6	24.7 6.4
3 degrees in roll angle	3.7 1.1	13.6 2.8	21.2 6.0	103.3 23.1
3 degrees in vertical angle	1.1 0.3	17.3 13.8	25.4 25.4	123.7 98.5
3 degrees in each angle	9.7 2.6	53.5 21.3	72.2 38.9	351.4 150.7

(1) The absolute values of the expected $\Delta\rho$ (or ρ or y) from an unperturbed scanner minus the $\Delta\rho$ (or ρ or y) from a scanner that has been rotated in the manner listed in the 'Perturbation' column.

(2) Maximum error in image.

(3) Maximum error in central 30 degrees of image.

Figure 3.5a Range images of an obstacle on a road

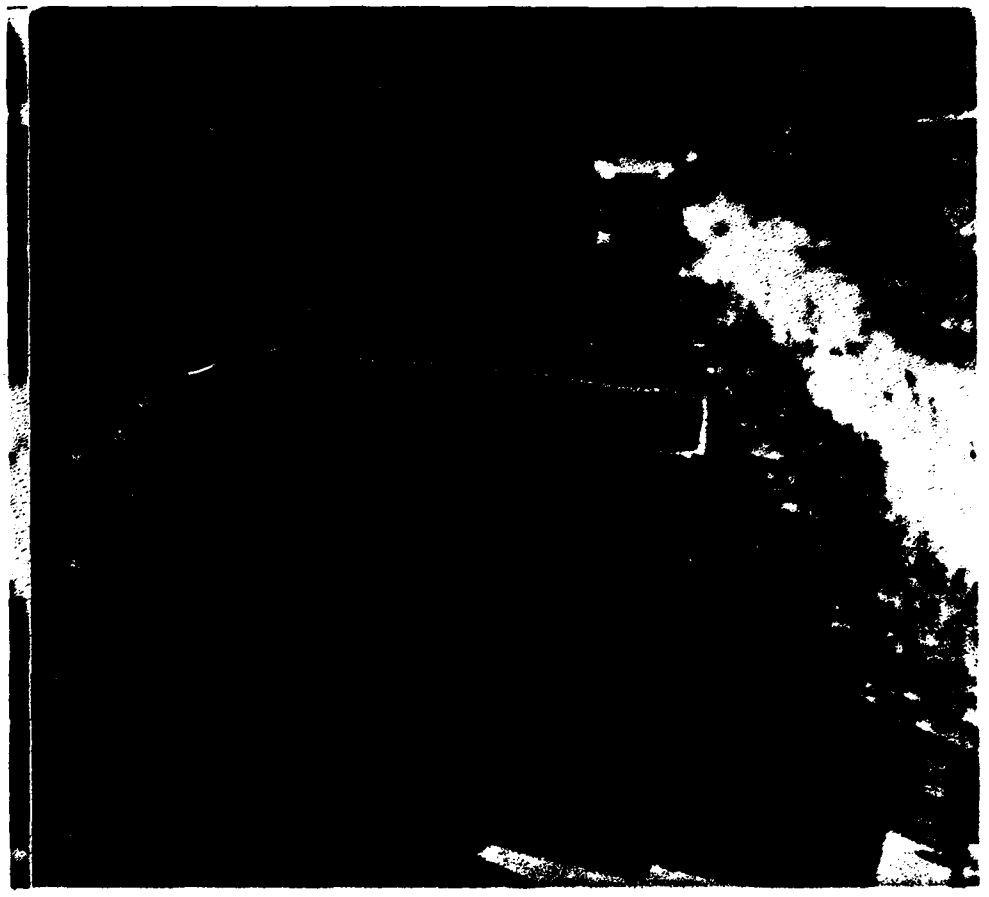


Figure 3.5b Video image of obstacles on roads

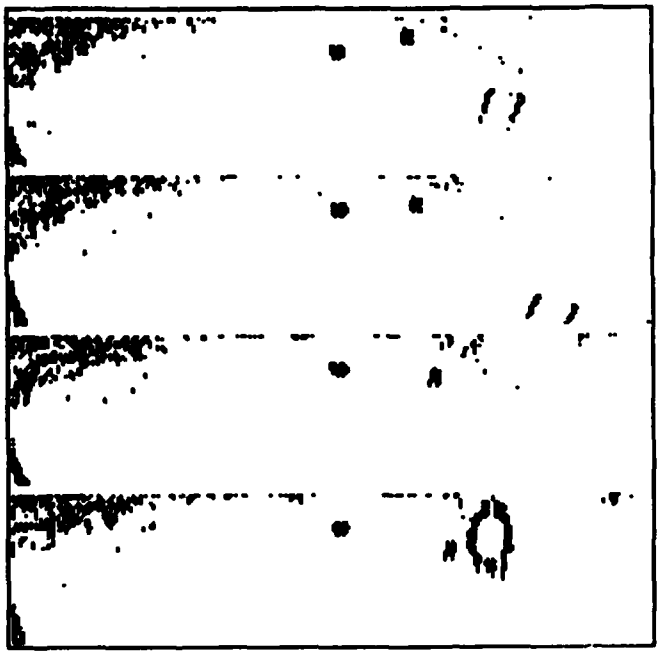


Figure 3.6a Thresholded θ derivatives

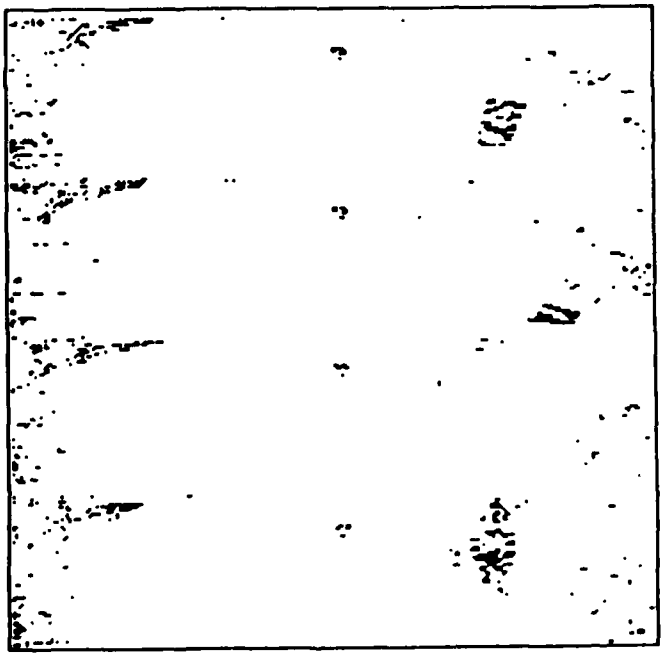


Figure 3.6b Thresholded ϕ derivatives of first montage

3.4 Geometry module

Consider the image of the road, which has been processed so that the curves which are images of the two road edges have been found (Figure 3.7). The geometry module reconstructs from these two image curves two world curves corresponding to the edges of the world road.

Consider a point p_i on one edge image. The world point of the road edge is a point P_i somewhere along the line Op_i, and the vector P_i is $m_i \boldsymbol{p}_i$; the parameter m_i is the unknown which, once found, will define the world point P_i.

Given a world point P_i on one world edge, there is a corresponding point P'_i on the other world edge such that $P_iP'_i$ determines a line normal to the road centerline. P'_i will be called the *opposite* point of P_i, and the segment $P_iP'_i$ a *cross-segment* of the road. The image p'_i of P'_i is somewhere on a road image edge (the one to which p_i does not belong). The position of $p'i$ can be defined on this edge in terms of a yet unknown curvilinear abscissa s_i ,

$$\boldsymbol{p}'_i = \boldsymbol{p}'_i(s_i),$$

and the world point P'_i which belongs to the line Op'_i can be defined by writing the vector $\boldsymbol{P}_i$ in terms of two parameters,

$$\boldsymbol{P}'_i = m'_i \boldsymbol{p}'_i(s_i),$$

where s_i defines the position of p'_i on the image edge, and m'_i defines the position of P'_i on the line Op'_i.

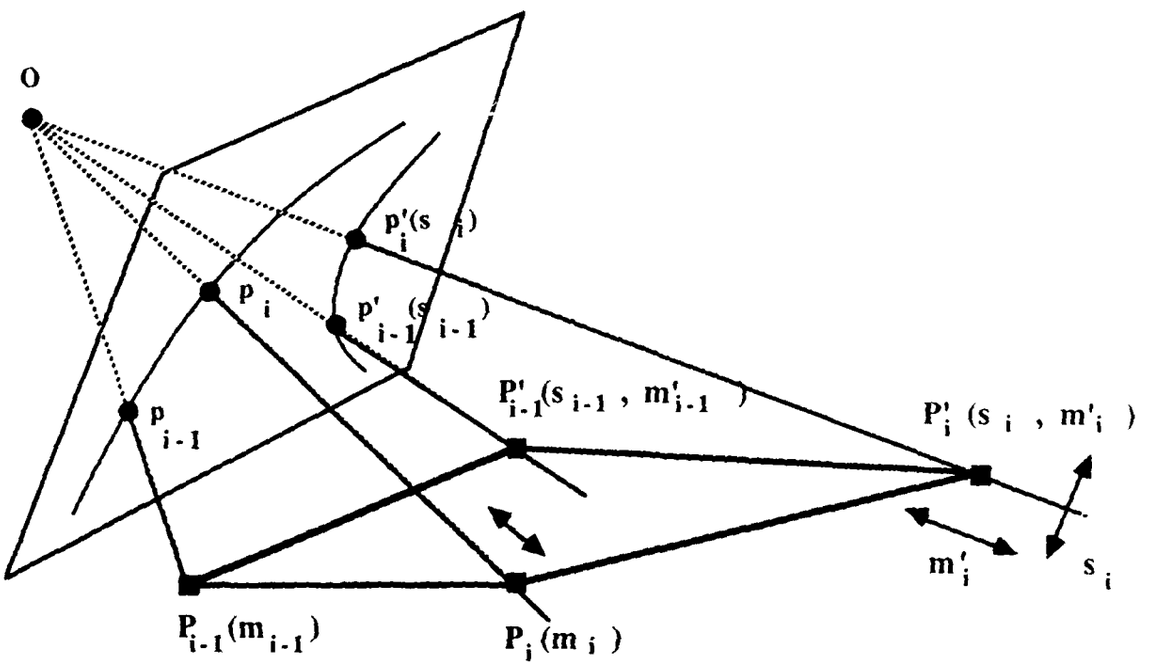


Figure 3.7 Method for the reconstruction of the road from an image

We can now consider two pairs of opposite world points (P_{i-1},P'_{i-1}) and (P_i,P'_i). Requiring the segments $P_{i-1}P'_{i-1}$ and $P_iP'_i$ to be cross-segments of the road is equivalent to saying that they are *normal to the centerline*, or that their average direction is normal to the line joining their mid-points. This is written

$$(\boldsymbol{P'}_{i-1}-\boldsymbol{P}_{i-1}+\boldsymbol{P'}_i-\boldsymbol{P}_i)(\boldsymbol{P}_{i-1}+\boldsymbol{P'}_{i-1}-\boldsymbol{P}_i-\boldsymbol{P'}_i) = 0$$

or

$$\boldsymbol{P}_{i-1}\boldsymbol{P'}_i^2 = \boldsymbol{P'}_{i-1}\boldsymbol{P}_i^2$$

In other words, *the two diagonals* of the convex quadrilateral formed by $P_{i-1}P'_{i-1}$ and $P_iP'_i$ *must be equal.*

Instead of imposing a strict equality, we can try to minimize the difference between the terms while keeping in mind that the points also have to satisfy other constraints:

$$E_{1i}=(\boldsymbol{P}_{i-1}\boldsymbol{P'}_i^2-\boldsymbol{P'}_{i-1}\boldsymbol{P}_i^2)^2$$

We also look for a road with an approximately constant width, and thus look for the points which minimize the quantity

$$E_{2i}=(\boldsymbol{P}_{i-1}\boldsymbol{P'}_{i-1}^2-\boldsymbol{P'}_i\boldsymbol{P}_i^2)^2$$

We could impose the further condition that the cross-segments be more or less horizontal, thus look for segments which minimize their scalar product with a vertical vector:

$$E_{3i}=(\boldsymbol{P'}_i\boldsymbol{P}_i\cdot\mathbf{V})^2$$

If P_{i-1} and P'_{i-1} are known from a previous calculation step, by setting E_{1i},E_{2i} and E_{3i} to zero three equations to calculate the three unknowns m_i,m'_i and s_i are obtained. A method of solving this problem ("the zero-bank algorithm") has been proposed (see DeMenthon [6]), for the case when the edge images are chains of line segments. m_i is found as a solution of a cubic equation, and an additional condition of limited slope variation of the road is required to choose among the solutions of this equation. Once the cross-segment P'_iP_i is found, the cross-segment $P'_{i+1}P_{i+1}$ can be determined. The method is an iterative reconstruction of the road, and therefore the first world segment must be known to start the process. Also, we can expect this method to deteriorate from error accumulation, and to perform poorly in case of bank or width variations of the world road. This algorithm is illustrated in Figure 3.8.

A potentially more robust approach would be to consider the global information contained in the image, and to construct the world road which minimizes the sum of the width and bank variations for segments normal to its centerline.

Consider the quantity

$$E = A\sum_{i=1}^{n} E_{1i} + B\sum_{i=1}^{n} E_{2i} + C\sum_{i=1}^{n} E_{3i}$$

A, B, C are weighing factors chosen to give more or less respective importance to the three conditions.

We could search for the parameters m_i,m'_i and S_i which minimize this function. One approach would be to set to zero the derivatives with respect to each of the parameters, and attempt to solve the $3n$ equations for the $3n$

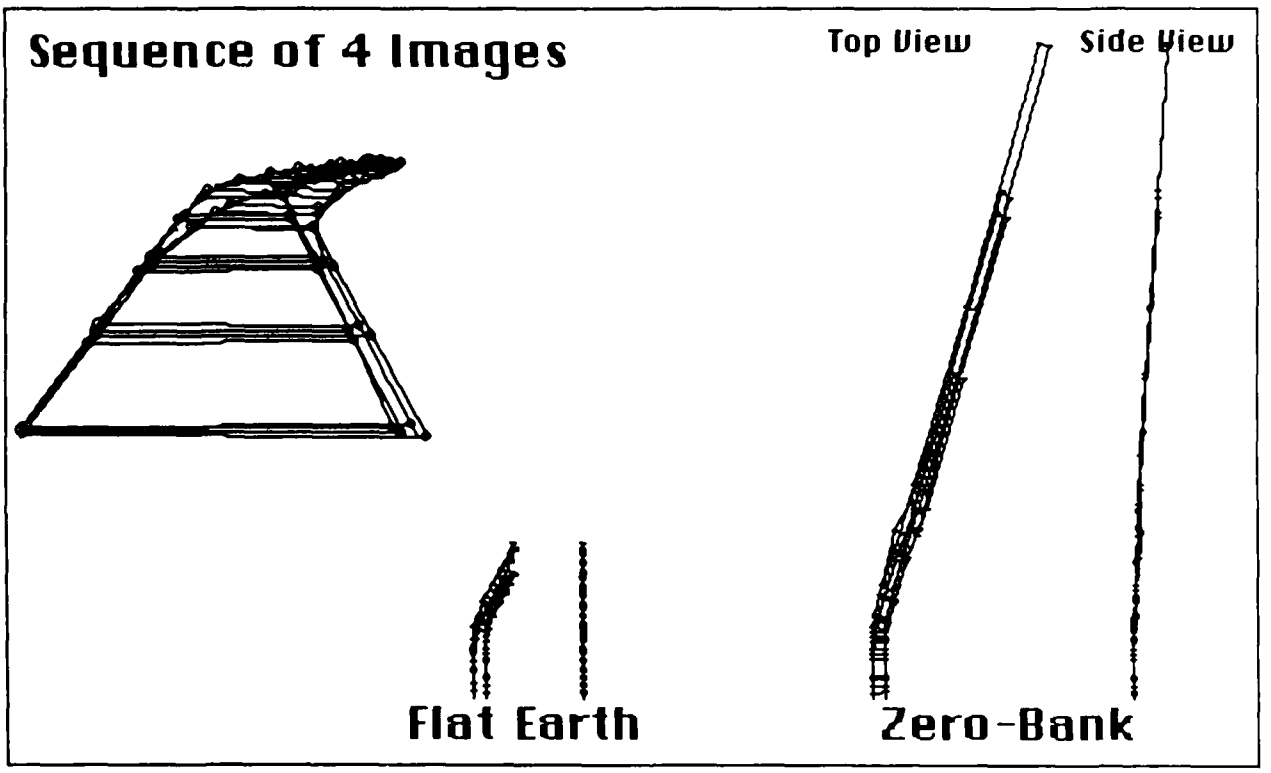


Figure 3.8 Left: sequence of four images of a curved road for four successive positions of ALV down the road
Middle: Reconstruction by flat earth assumption.
Right: top view and side view of 3-D reconstructions by zero bank algorithm. The four reconstructions are superimposed in a global coordinate reference.

unknowns. Unfortunately, the equations are nonlinear equations of the third degree. A more promising method being presently developed consists of having the different parameters *bid* for the right to be incremented or decremented, the level of the bid being a function of the promise of a resulting decrease of the global "energy" term E, and also a function of their strength; the strength results from *rewards* from the neighbor parameters when they won bids following a move of the considered parameter; the strength reinforces the positive nonlinear interactions between parameters. This method can be run by parallel processors, each processor being assigned one parameter.

3.5 Map database

A map database of the ALV test site has been constructed by the U.S. Army Engineer Topographic Laboratories. This map contains information about roads, topography, surface drainage and landcover relative to the test site. We have constructed a similar map of our terrain board which contains only information about the network of roads. The map contains information about road width, road composition (asphalt, concrete, etc.), road markings (e.g., lane markers), boundary information (presence of road shoulders, properties of the surrounding terrain—e.g., vegetation, sand, etc.) and network topology and geometry. The roads are segmented into pieces which have (nearly) constant properties. So, for example, if a road changes width at some point then that point would be the boundary between two road pieces in the representation of the road. Of course, it is not necessarily the case that all of this information would be provided, a priori, in the map database. Some of it may be acquired as the vehicle travels along a road, and some may simply not be available to the vision executive in planning its strategy for identifying the road.

The map is preprocessed by computing, for a regular sampling of points along the roads to be traversed in a mission, properties of the expected appearance of the road network when viewed from approximately those positions. These predictions are used by the vision executive both as a source of guidance for planning the image processing operations to be applied to images and as a source of constraints on the interpretation of those analyses. For example, if it is known that the road has a bright centerline and is on relatively flat terrain, then

the vision executive might plan to detect the road by searching for that center-line and to reconstruct the three dimensional structure of the road based on a priori estimates of road width and the flat earth model. In Section 7.0 we present an example in which the vision executive uses precomputed map predictions about the appearance of an intersection to control the processing of an image of the intersection and to navigate the robot arm through the intersection.

3.6 Predictor

Section 3.2 discussed the image-processing strategy applied by the feed-forward algorithm: the image analysis is limited to a succession of windows, the position of a new window being deduced from the road edge element detected in the previous window.

The question which the predictor answers is: where do we place the first windows in the image? The basic idea is to position the first windows by predicting approximately where the road edges are going to be in the image, using the images of the edges detected in a previous image. This procedure is illustrated in Figure 3.9. There the image plane of the vehicle is shown as a line normal to the camera axis and in front of the lens centers, and the camera moves from position C_1 to position C_2. The change of position of the camera from C_1 to C_2 is available from the navigation system of the vehicle. Consider the image m_1 of a world point M for the camera position C_1. Where is the image m_2 of this point M for the camera position C_2?

The drawing shows the method: from m_1 we must deduce the world point M by an *inverse perspective* method. Figure 3.9 shows a simple flat earth assumption in which the ground is assumed planar. M is then the central projection of m_1 on the ground plane. Note that the whole figure is determined entirely by the following elements:

1. the ground, defined by its distance h from the camera positions and by a normal **n**;

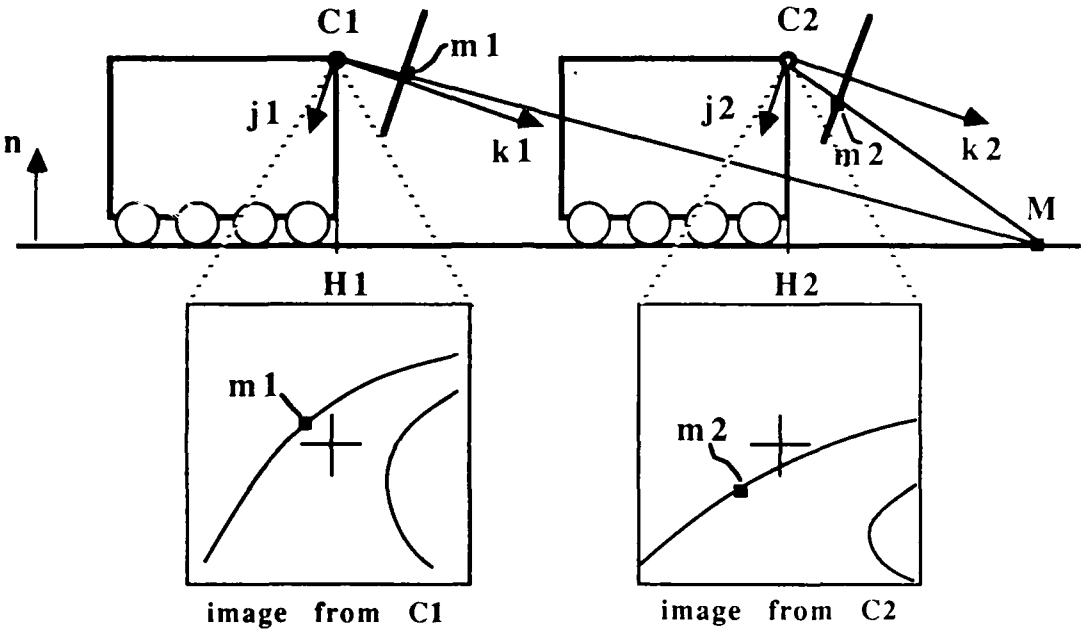


Figure 3.9 Basic principle of prediction with flat eath assumption

2. the image m_1, defined by a vector C_1m_1;
3. the camera focal length f;
4. the new camera position C_2, defined by the translation vector C_1C_2 and the coordinates of the unit vectors of its reference (i_2,J_2,k_2) in the coordinate system of the first camera position C_1.

The unknown is the vector C_2m_2, easily expressed in terms of the known elements. The fact that M is the projection of m_1 in the ground plane determines the vector C_1M:

$$C_1M = a_1C_1m_1,$$

$$C_1M.n = h,$$

$$a_1 = h/C_1m_1.n,$$

$$C_1M = hC_1m_1/C_1m_1.n$$

C_2M is now known:

$$C_2M = C_1M - C_1C_2,$$

m_2 is the projection of M on the image plane at position 2:

$$C_2m_2 = a_2C_2M,$$

$$C_2m_2.k_2 = f,$$

thus:

$$a_2 = f/C_2M.k_2$$

$$C_2m_2 = fC_2M/C_2M \cdot k_2$$

Of course, all the vectors have to be expressed in the same reference frame, for instance the reference frame of C_1. The coordinates of m_2 in the second camera position C_2 are then the scalar products of C_2m_2 in the reference frame of C_1 with the unit vectors of the reference of C_2 expressed in the reference of C_1, i.e., $C_2m_2.i_2$ and $C_2m_2.j_2$.

It would therefore be possible to take all the significant points of image 1 and transform them to points of image 2. However, since the vehicle moved between C_1 and C_2, most of the world points corresponding to the C_1 image points will not fall within the field of view of C_2. Furthermore, all we require are two edge points in the image of C_2 on which to anchor the bottom midpoints of the first processing windows. Note that for the windows to fit in the image rectangle, these anchor points must belong to a *reduced rectangle*, obtained by reducing the image rectangle laterally by half the window width, and at the top by the window height, as shown on Figure 3.10b. The simplest approach to find the anchor points would then be to find the intersections j and j' of the predicted road edges with the reduced rectangle (Figure 3.10b).

This method consists of transforming segments $a_1b_1, b_1c_1, \ldots$ of C_1 (Figure 3.10a) into a_2b_2, b_2c_2 of C_2 (Figure 3.10b) until we find an intersection with one of the lower three sides of the reduced rectangles. If an intersection is found for only one edge, the other edge being totally out of the rectangle, a prediction can be computed for only one edge, and the vision executive is informed of this situation.

We will refer to this previous method as *method A*. It is clearly not optimal, because it places windows at the lowest possible point in the image; the vehicle might not need to know these lowest parts of the edges, because when the processing is over, the vehicle will probably have run *beyond* the corresponding world edges and have no more use for the information. In *method B* described next, we try to position the two anchor points at a higher position; however, if

method B fails, as happens when the proposed anchor points fall outside of the reduced rectangle, the system uses method A.

Method B is illustrated in Figure 3.10d. The drawing represents a top view of the world with two vehicle positions, giving camera positions C_1 and C_2. When the camera reaches C_2, its position with respect to C_1 can be calculated from the output of the navigation system. A plane normal to the vehicle direction at C_2, at a distance H_d in front of the vehicle is taken. The edge segments $a_1b_1, b_1c_1, c_1d_1, \ldots$, from the image at position C_1 are projected one by one to the ground plane, giving world edge segments AB, BC, CD, ... until a projection segment (such as CD in the figure) is found to cut the plane. As soon as an intersection I is found, its image point i is obtained in the image plane of C_2 (Figure 3.10d). If this image point i does not fall inside the reduced rectangle of the image, the method is stopped and method A, described above, is initiated to find a replacement anchor point. If, on the other hand, the image point i falls inside the reduced rectangle, it is chosen as an anchor point of the first window. This process is repeated for the images of both edges at position C_1.

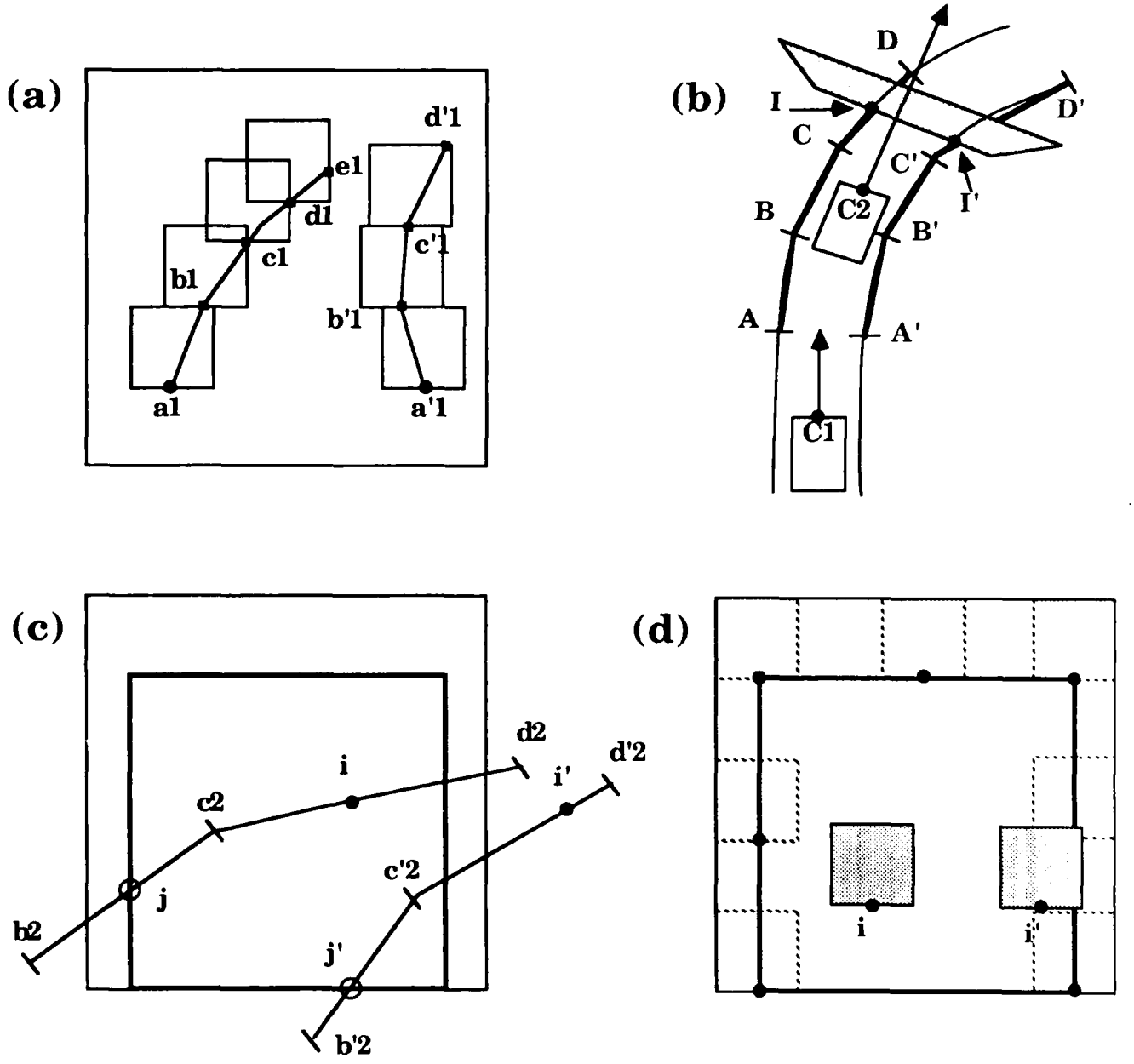


Figure 3.10 (a) Image from C1
(b) Top view of the world
(c) Image from C2: points j and j' are intersection of reduced rectangle with images of road edges (Method A)
(d) Image from C2: reduced rectangle and anchor points i and i' images of world points I and I' (Method B)

3.7 Sensor Control

The sensor control module determines pan and tilt angles for the television camera that will approximately center a particular roadmark in the next image acquired. The roadmark is chosen by the vision executive based on the speed of the vehicle and the length of the three dimensional road model constructed from analysis of previous frames. The camera is first tilted until a horizontal line passing through the roadmark projects onto the center horizontal scan line of the image. The camera is then panned until the roadmark itself projects onto the central column of the image. This simple procedure assumes that the relative roll angle between the camera and the cross section of the road containing the roadmark is zero. If this is not the case, then the camera can be first rolled by an angle that would result in no relative roll angle.

3.8 Representation

The vision executive sends to the representation module the sequence of surface patches for the current road. Associated with each patch is the following information:

(1) Four coefficients of the plane (or "surface-patch")
(2) Number of lanes
(3) 3-D coordinates of the two end-points of the left shoulder boundary
(4) Confidence in the left shoulder segment
(5) 3-D coordinates of the two end-points of the left road boundary
(6) Confidence in the left road segment
(7) 3-D coordinates of the two end-points of the right road boundary
(8) Confidence in the right road segment
(9) 3-D coordinates of the two end-points of the right shoulder boundary
(10) Confidence in the right shoulder segment.

If there are intersecting roads in the image, then for each intersecting road the following information is sent to the representation module by the vision executive:

(1) Closest patch on the current road
(2) Number of patches on the intersecting road
(3) For each patch:
    a) Four coefficients of the equation of the plane
    b) Number of lanes
    c) 3-D coordinates of the four border segments (road and shoulders borders) along with a confidence measure for each of these segments.

From this data, computed in the camera coordinate system, the centerlines of the current and intersecting roads are extracted, and roadmarks are placed along these centerlines at points of significant changes. For example, a new roadmark is computed each time a new surface patch has been found or when one of

the five parameters:

(1) Change in distance to the next roadmark
(2) Changes in road orientation
(3) Road bank
(4) Road slope
(5) Road width

change significantly. Each roadmark is an eight-element vector whose coordinates represen both relative information from one roadmark to the next and road information at the location of this roadmark.

Such a set of roadmarks is computed for the current road and for the intersecting roads. But, as is described in Section 3.2, image processing is not performed in the area of the intersection itself. Therefore, we have to extrapolate between the two sets of roadmarks to build a complete model of the intersection. An extrapolated roadmark is placed at the junction between the two roads and is stored as the corresponding roadmark of the intersecting road on the current road.

4. Path Planning

Given a robot's initial and final goal locations, the problem of mobile robot navigation is to find an appropriate path from an initial position to the goal position such that the mobile robot can smoothly travel through the area avoiding obstacles.

A mobile robot navigates with a limited knowledge of its environment because of restricted field of view and range of its sensors, and because of occlusion of parts of the environment in any single image. As a result the problem of reaching a final goal point has to be treated as a sequence of local problems. This will involve generation of subgoals and planning local paths to this sequence of subgoals which have to be attained successively in order to reach the final goal.

A mobile robot's path planning space can be decomposed into free regions, obstacles, occluded regions and unexplored regions. Most path planning algorithms consider only free regions and obstacles in the robot's world for path planning. The objective of our path planning algorithm is to extend the classical path planning paradigm to include occluded regions; this introduces the new problem of deciding when (or whether) to employ the vision system during the execution of the path to, potentially, reveal the occluded regions as obstacles or free space for the purpose of replanning.

We have assumed that the robot's path planning space is modeled by the cartesian plane containing polygonal obstacles. A height specification is associated with each polygon for outlining the occluded regions in the robot's world. Simu-

lation of the robot's vision system is performed under constraints of field of view and range of vision for the purpose of classifying the robot's environment into free, obstacle, occluded and unexplored regions.

Section 4.1 describes an optimistic path planner which uses the multiresolution representation based path planning algorithm described in Kambhampati and Davis [7] to plan a negotiable path. A cost function is developed which estimates how close to optimal this negotiable path is. A decision procedure is employed to decide how far to travel along a path before employing the vision sensor module again to potentially map the occluded regions in the shadows of obstacles in the hope of discovering a lower cost path by replanning.

Section 4.2 gives results of the implementation of vision simulation and the path planner.

4.1 Multiresolution representation based path planning

A number of algorithms for mobile path planning based on a multiresolution representation of the robot's immediate environment are presented in [7]. The multiresolution representation used is the quadtree (Samet [8]). Free regions are represented by white nodes and obstacle regions by black nodes.

The path planning algorithm is a simple A^* search algorithm with the evaluation function, f, defined as follows :

$$f = g + h$$

Here g is the distance of the current node in the search from the start node, and h is a heuristic estimate of the goodness of the remainder of the path P. The

heuristic h is the difference of two components, h_G and h_d, where h_d is the distance of the nearest obstacle from the current node, which determines by how far the resultant path will avoid obstacles, and h_G is the straight line distance between the current node and the goal. The result of applying the A^* algorithm to the quadtree is a list of nodes from the quadtree which define a path between the start node and the goal.

Staged search is an important extension of this simple path planning algorithm involving the ability to deal with grey nodes in the quadtree (nonterminal nodes which have both black and white descendants). Dealing with grey nodes can greatly reduce the number of blocks that need to be considered in building an initial estimate of a path, resulting in computational savings.

In Puri and Davis [9], the impact of revealing occluded regions in the robot's world by sensing at a point along this path is additionally considered to decide whether replanning would be cost-effective. This algorithm is called the *optimistic path planner*. The optimistic path planner will reveal occluded regions in the vicinity of the planned path as it traverses the path in the hope of discovering a lower cost path to the goal.

The cost of a path can be measured in terms of the energy expended by the mobile robot in traversing that path. It will depend, among other things, on two factors the length of the path and number of direction changes in the path. The smaller the distance traveled and the fewer the direction changes made by the robot, the lesser will be the time and energy used by the robot in traversing the

path. Hence, a straight line path between the start and goal points would be the optimal path. The goal is to reveal occluded regions in the vicinity of the negotiable path to develop a path which is closer to a straight line path.

A path is said to be *acceptable* if no replanning is required for it. Let l_1 be the length of a path planned under the assumption that all occluded regions in the world are actually obstacles, and let l_2 be the length of the path planned under the assumption that all occluded regions are actually free regions.

The ratio l_2/l_1 is a measure of the impact of the occluded space on the negotiable path. If the ratio is very close to 1, then revealing the occluded regions would not help in finding a lower cost path. But calculating the ratio l_2/l_1 would require running the local path planner twice. To avoid this extra computation, consider the length, l_3, of the straight line path from the start point to the goal point in question. The straight line path is the least expensive path that can be traversed by the robot to reach the goal point. The straight line path between the start point and the goal point may not exist in reality but the ratio l_3/l_1 can give us a heuristic measure of acceptability of the path. This ratio is always less than or equal to 1. If l_3/l_1 is very near to 1, the path P is very close to the straight line path between the start and goal points. Replanning by revealing the occluded regions in the vicinity of the path P will not be worth the cost. The path P is acceptable. If l_3/l_1 is close to 0, revealing occluded regions in the vicinity of the path P may help in finding a lower cost path.

Once it has been decided that the path P given by the local path planner is not acceptable, a decision has to be made concerning from which point on P to use the vision system. P will be a sequence of points $\{p_1, p_2, \ldots, p_n\}$ specifying the quadtree nodes constituting the path. The decision procedure will depend on two factors—(1) the amount of occluded space that can be potentially revealed from a particular point on the path and (2) the distance to be traveled to reach that path point.

The next picture should be taken from a point where a large amount of occluded space is potentially revealed and a small distance is traveled to reach it. Let $V(p_i)$ denote the area of occluded space visible from the path point p_i. The decision can be based on a gain function computed for each path point, which will be directly proportional to the visibility measure of a path point, $V(p_i)$ and inversely proportional to the distance traveled, $S(p_i)$.

Our current decision procedure analyzes the shape of the derivative of the visibility measure, $V(p_i)$, with respect to distance traveled along the path, at the discrete set of path points given by coordinates of quadtree nodes constituting the path. Thresholding this derivative at some appropriate value will give the path points where the visibility is increasing at a sufficient rate. Starting at the first path point, we calculate the visibility measure for successive path points until we find a path point whose visibility derivative is above the threshold. We continue with the calculation for successive path points while the derivative is above the threshold. The last path point for which the derivative is above the threshold is the point at which we take the next picture of the world to expose

occluded regions. Since the total amount of occluded space is constant in a single image, if we keep the threshold for the derivative at a relatively low value, the best candidate path point found will be close to the start point.

An estimate of visibility measure i.e., the number of pixels of occluded space that may get exposed from p_i, can be computed by analyzing the line of sight from p_i to each of the constituent points of occluded space in the robot's map of the environment. If, for any such point, this line of sight does not intersect an obstacle or another occluded region then we assume that this point of occluded space is visible from p_i. This computation is done under the assumption that occluded space hides a free region.

In simple cases of path planning, it can be assumed that backtracking is unnecessary. Hence, the visibility measure of a path point can be computed only for those constituents of occluded regions which lie within an 180° field of view of the path point under consideration, and viewing in the direction of the goal. Also, the constituents of occluded regions considered for this purpose must be within the range of the sensor.

4.2 Experimental results

Experiments were performed on a 64×64 pixel image map containing 25 obstacles. With a vision sensor range of 50 pixels, the vision sensor simulation took typically one second to run on a VAX - 11/785 running 4.3BSD Unix with five obstacles in its range of vision. The simulator program was written in C.

Figures 4.1 and 4.2 show the result of planning and improving a path using the optimistic path planner on paths returned by the staged path planning algorithm of [7]. In the figures, free space is shown by white areas, obstacles are black areas, occluded regions are indicated by light cross-hatching and unexplored regions by dots. The path is shown by dark cross-hatching and the start and goal nodes in the quadtree by single-hatching. Time taken in seconds by the staged path planner is given below each figure. Replanning was performed in the third large block of the quadtree path.

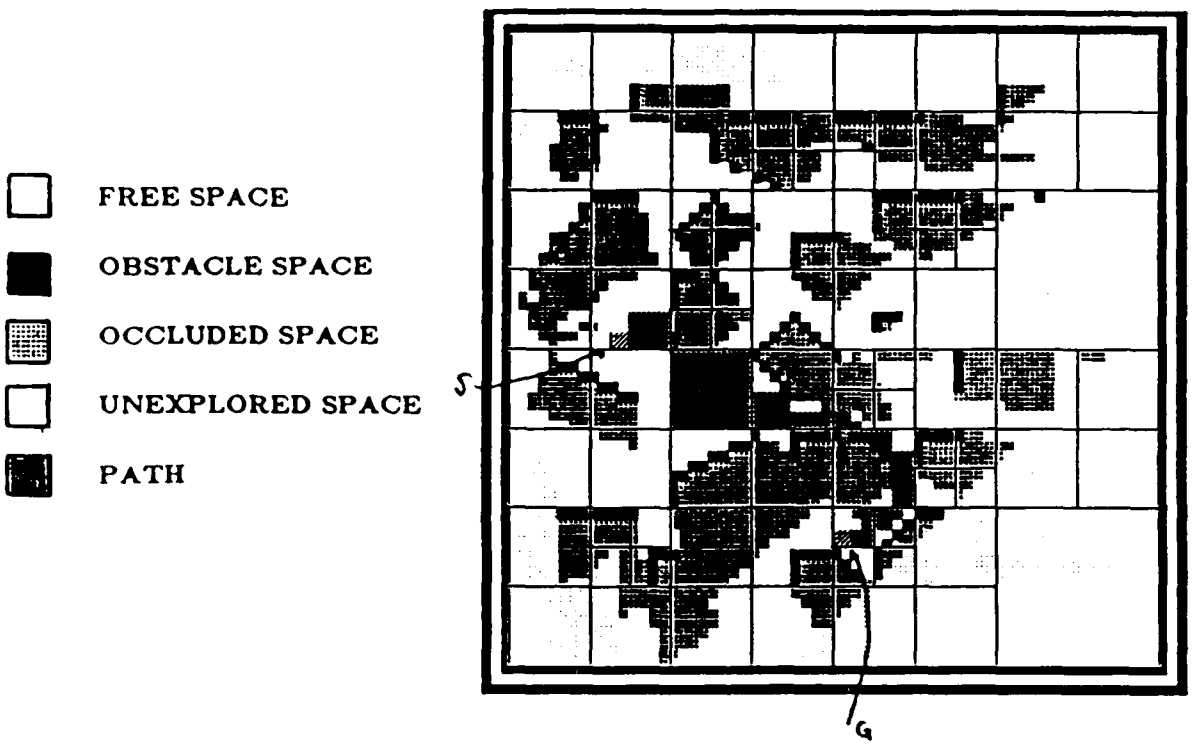


Figure 4.1 A path is planned assuming that all the occluded space is obstacles. The value of the cost function is found to be below an acceptable threshold.

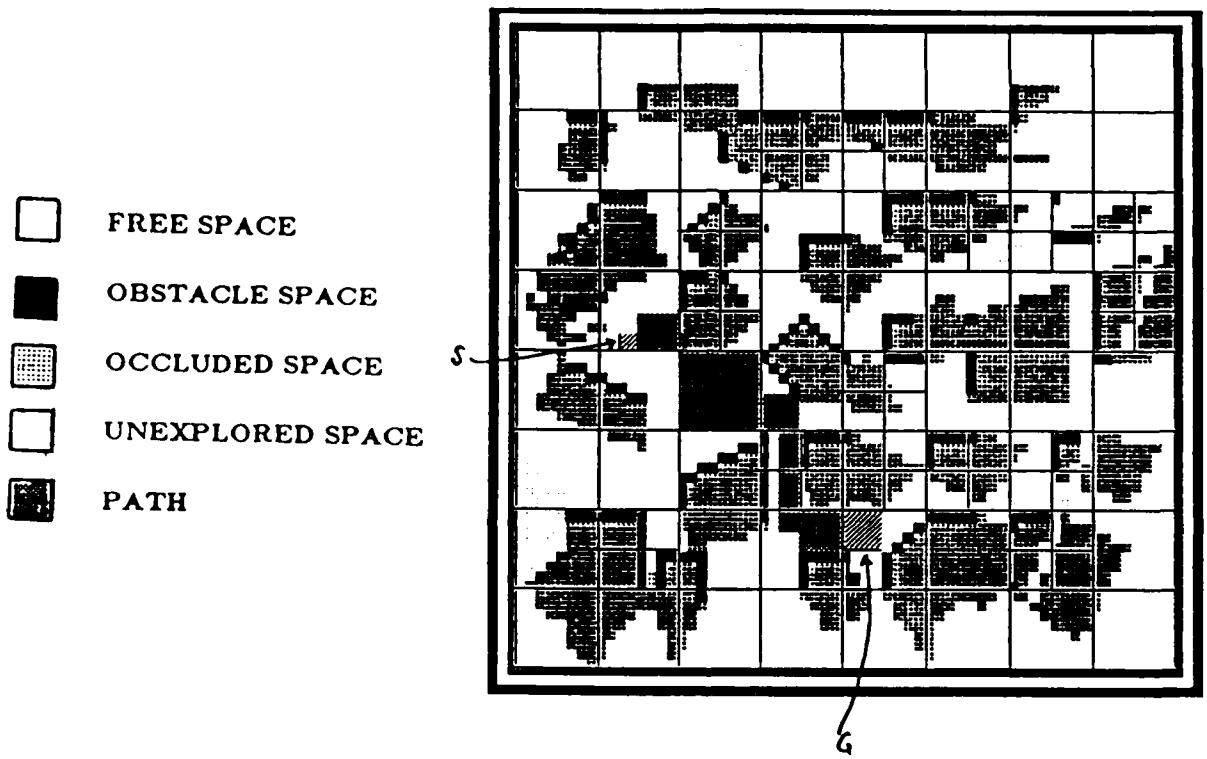


Figure 4.2 A new path is planned from the best candidate path point tc the best candidate path point to the goal point after taking new pictures from the best candidate path point.

5. Rule-Based Visual Navigation

The scene model is the central component in the ALV vision system and represents the vehicle's interpretation of its environment. Construction of the scene model is complex, requiring the direction of vehicle sensors towards objects in the world, the fusing of data from different sensors, and the selection of algorithms for image analysis. Methods for performing these tasks are continually evolving as the ALV road following task becomes better understood. Thus, the control structure of a system for constructing a scene model must be flexible enough to accommodate new strategies for performing these tasks. This section describes a new approach to scene model construction which satisfies this requirement and offers other advantages as well.

The task of building a scene model for the ALV involves two major subtasks: 1) deciding what objects to look for and where, and 2) verifying that the objects exist in the world. These two functions are performed by the Scene Model Planner and Scene Model Verifier, respectively; together, they form the Scene Model Builder. The data flow diagram for the Scene Model Builder is presented in Figure 5.1. The scope of the current research is the design and implementation of the Scene Model Verifier; the Scene Model Planner will be simulated by a human operator. Thus, the Scene Model Verifier will be passed an object hypothesis and will return the object along with a measure of its verifiability. If the object cannot be verified, this confidence level will be very low.

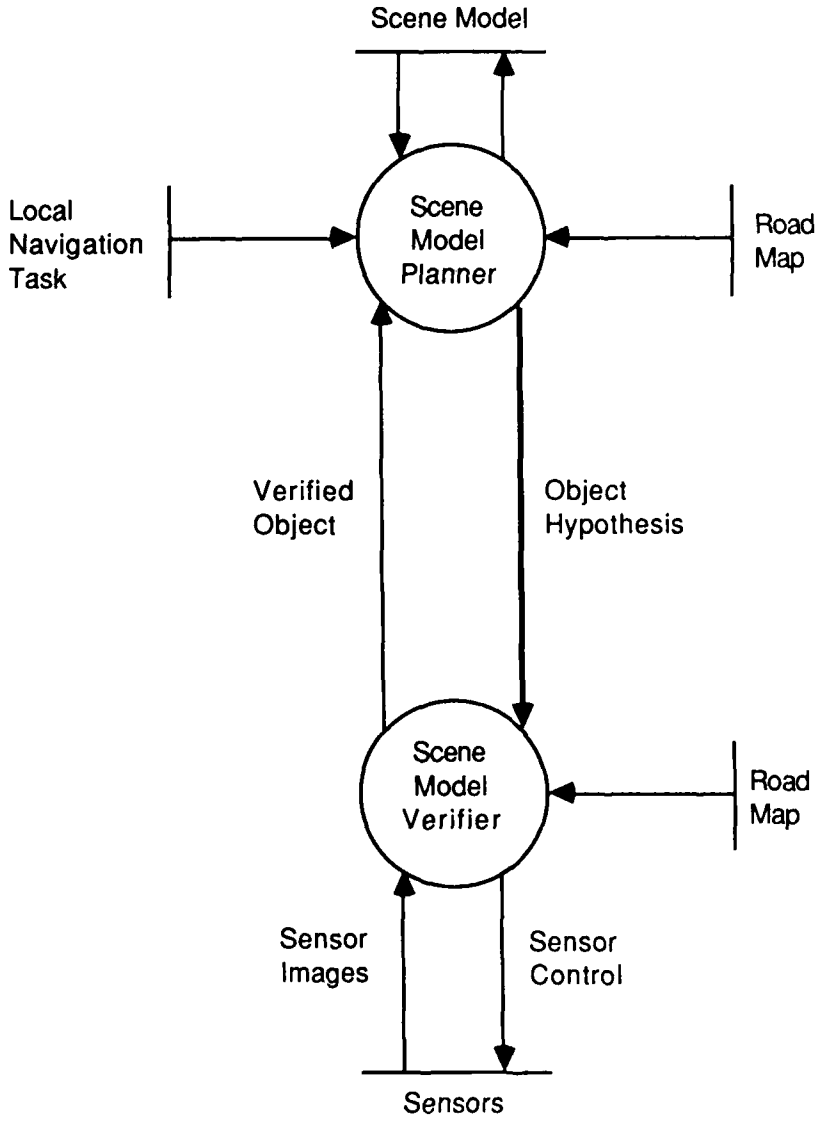


Figure 5.1 Scene model builder dataflow

Objects in the world exhibit the following relationships:

- *Component Relationships* For example, an intersection is made up of four connecting roads, stop lights, etc. These component objects, in turn, can be decomposed into their component objects, e.g. a road is made up of two road edges, two shoulder edges, and a centerline. However, primitive objects such as lines and surfaces extracted from an image cannot be decomposed.
- *Spatial Relationships* For example, telephone poles are located near the road and run parallel to the road.
- *Property Inheritance* For example, although the door to a house and the door to an automobile may be different objects, they both share the generic properties of a door, e.g. handle, dimensions, and direction of opening.

To accommodate these three relationships, frames have been chosen to model objects. A frame is a data structure which encapsulates the relevant knowledge about an object. This knowledge is stored in a set of slots (or attributes) assigned to the frame. Slots may contain values, e.g. the width of a road patch, or pointers to other frames, e.g. component and spatially related frames. Property inheritance among frames is accomplished by supplementing a frame's slots with the inherited frame's slots. Figure 5.2 presents an example of two frames representing road patch and planar ribbon objects. The road patch frame inherits the slots of a planar ribbon frame. Through this inheritance, the road patch frame includes slots pointing to left and right segment component frames as well as front and back connected planar ribbon frames. In this example, the

Road Patch Frame Definition

Frame Class:

road patch

Frame Attributes:

type of left world segment
type of right world segment
width confidence

Inherited Frames:

planar ribbon

Planar Ribbon Frame Definition

Frame Class:

planar ribbon

Frame Attributes:

search window
back connected planar ribbon
front connected planar ribbon
has part left world segment
has part right world segment
parallelism confidence
confidence

Inherited Frames:

none

Figure 5.2 Road patch and planar ribbon frames

road patch is considered to be a specialization of a planar ribbon.

In the context of the scene model builder, a hypothesis is a frame whose slots have not been completely filled. Verifying an object hypothesis, i.e. accumulating evidence supporting the object's presence in the world, amounts to filling in the slots of the object's frame with meaningful values. The mechanism for filling in these slots is provided by a blackboard.

The blackboard can be viewed as a small production system whose facts point to object frames, and whose rule-base contains the rules by which frame slots are filled. To improve efficiency in rule firing and to promote modularity, a separate blackboard exists for each class of object. As with frames, the blackboards exhibit component, spatial, and inheritance relationships.

When an object's class and location can be predicted with sufficient confidence, top-down hypothesis generation is invoked to verify the prediction. If the object can be sufficiently verified, the scene model planner adds it to the scene model. Figure 5.3 presents an example of top-down hypothesis generation used to verify a road patch hypothesis. To begin the process, a road patch frame is instantiated, some of its slots are filled in, and it is placed on the road patch blackboard. When a new object appears on the blackboard, rules responding to the empty slots invoke knowledge sources to fill in the slots. These activated knowledge sources have access to knowledge contained in the object's frame and any frame directly or indirectly connected to the object's frame. When an empty slot represents a relationship to another frame, e.g. component, a new frame is instantiated on the blackboard representing the class of the related frame.

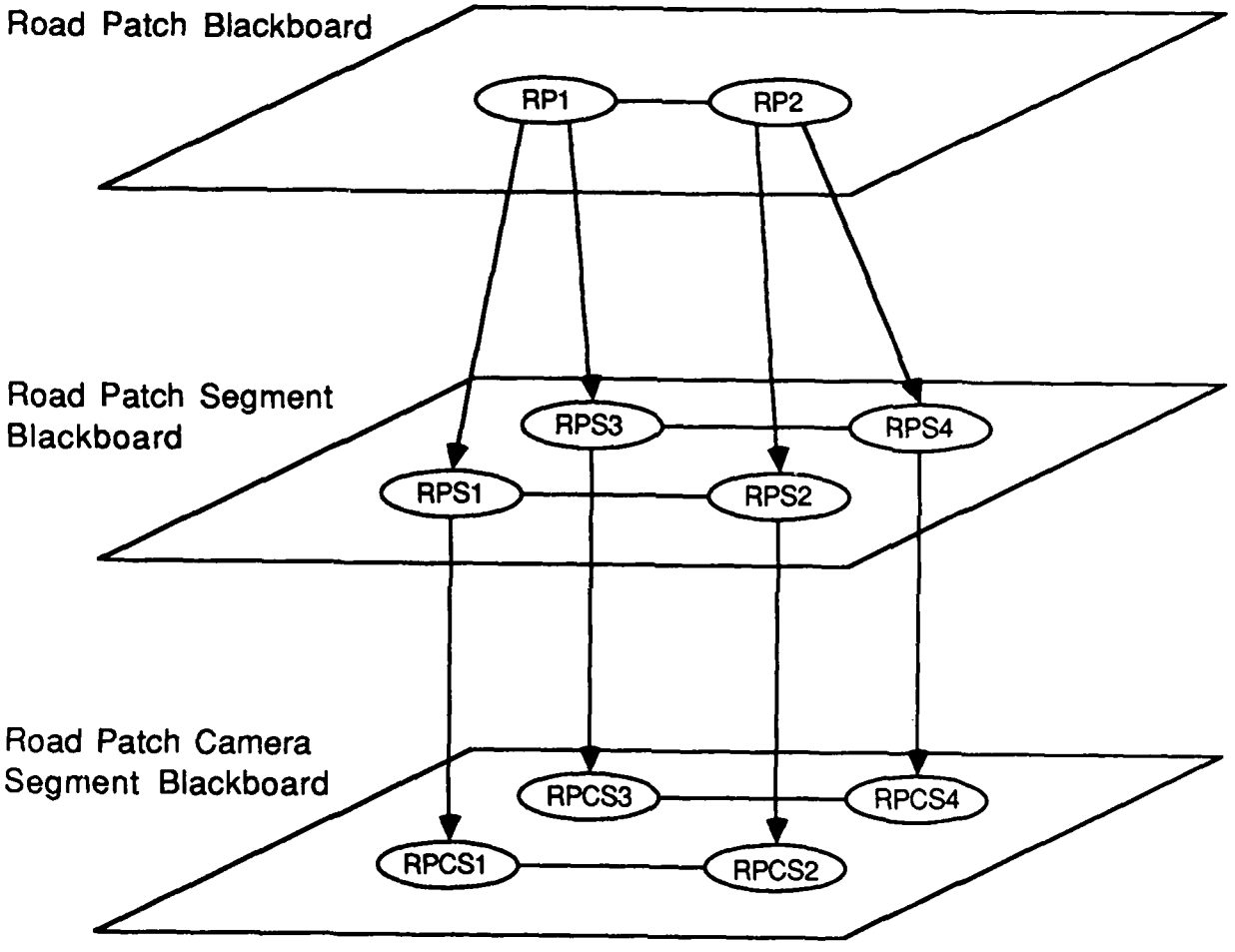


Figure 5.3 Top-down hypothesis generation

Again, rules fire automatically to fill in the empty slots of the new frame. Eventually, the filling of slots of a primitive object frame requires the extraction of features from an image.

In top-down hypothesis generation, hypotheses deposited on upper blackboards generate hypotheses on lower blackboards. The last slot in each frame to be filled is the confidence of the frame. It is computed based on the values of the other slots as well as values of slots in connected frames. After hypotheses are pushed down the blackboard hierarchy, the confidences are bubbled up the hierarchy. When the confidence of the initially instantiated object is determined, the process terminates. All that remains on the blackboards are the verified objects.

When the scene model planner has little or no knowledge about the vehicle's environment, bottom-up hypothesis generation may be invoked to locate any objects of interest in the image. To begin the process, primitive features are extracted from the image; for each feature, a road patch camera segment frame is deposited on its blackboard. Since the "part-of" slots of these frames are unfilled, the rules generate hypotheses representing the possible objects of which these primitive frames may be a component. The process continues, pushing hypotheses up the hierarchy. In Figure 5.4, four hypotheses appear on the road patch blackboard. It is up to the scene model planner to decide whether these objects meet the necessary criteria for insertion into the scene model.

The Scene Model Verifier is object oriented and runs in the Franz Lisp environment; both the frames and the blackboards are implemented using Franz

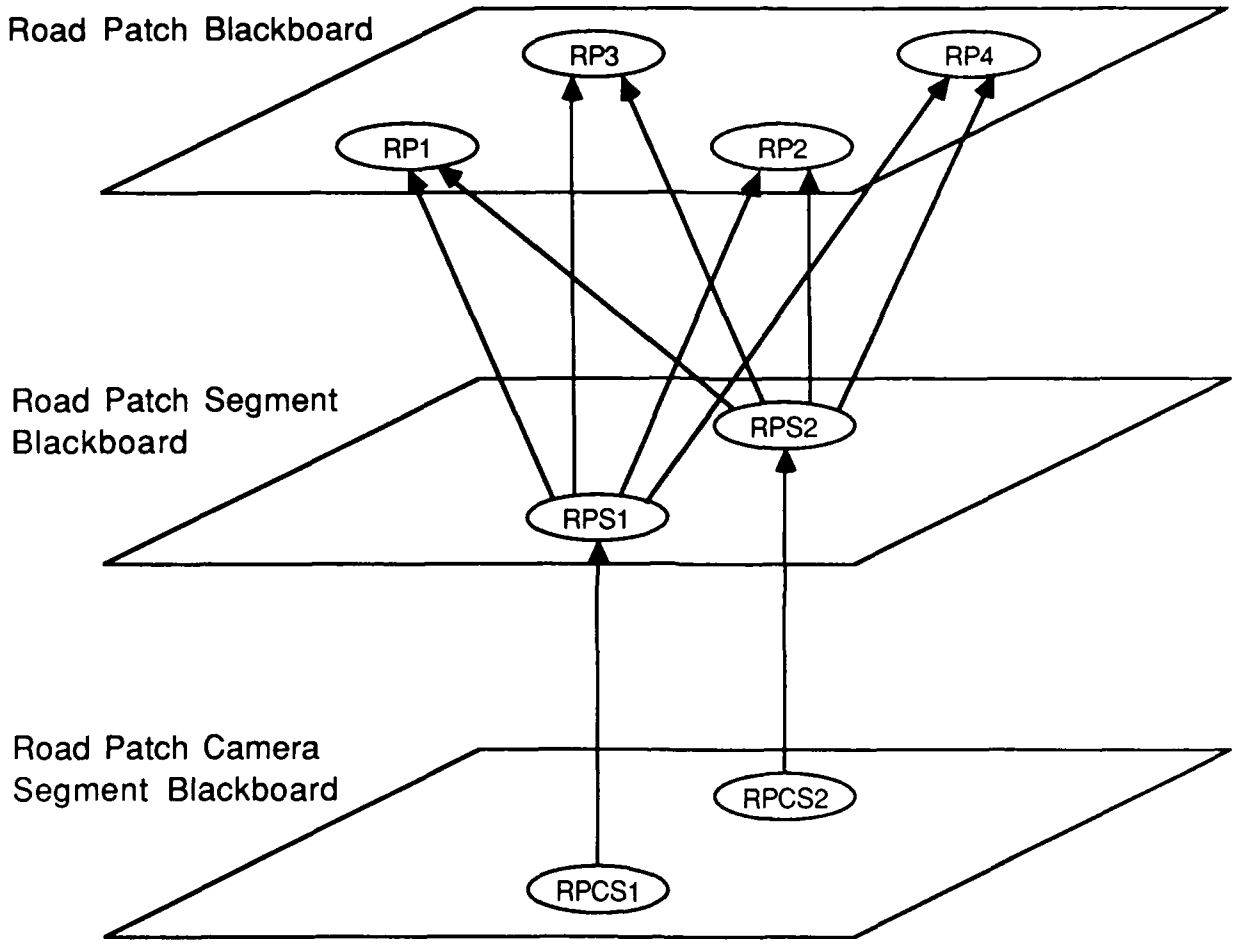


Figure 5.4 Bottom-up hypothesis generation

Flavors. In addition to slots referencing external entities such as the road map and the scene model, each blackboard object has its own working memory and rule memory. Rules may invoke Lisp functions which, in turn, may invoke C functions to fill a frame's slots.

6. Butterfly Algorithms

The Computer Vision Laboratory has two Butterfly parallel processors; one is a 16 processor machine, with MC68020 processors and 64Mbytes of memory, and the second is a 128 processor machine with MC68000 processors and 128Mbytes of memory. In Section 6.1 we present a brief overview of the organization of the Butterfly.

We have used the Butterfly as an experimental machine for research in parallel vision and AI algorithms. In Section 6.2 we discuss parallel algorithms for computing Hough transforms, and describe the results of experiments with those algorithms on the Butterfly. The Hough transform programs were written in C using the Uniform System. They have since been integrated into a larger system of programs that include most of the algorithms comprising the feedforward mode of road boundary detection. This larger system is described in Puri and Davis[10]. We hope to be able to perform experiments with this system either on the ALV or on recorded data from the Martin Marietta test site during the coming year.

We have also been studying parallel search algorithms using the Butterfly LISP (BLISP) system. This system is based on the MultiLISP extension of LISP developed by Halstead[11] at M.I.T. In Section 6.3 we describe this research.

6.1 The Butterfly parallel processor

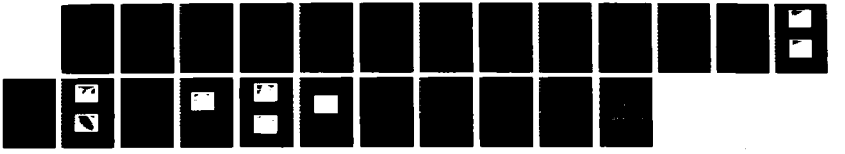
In this section, we review the Butterfly architecture. For more information, see [3].

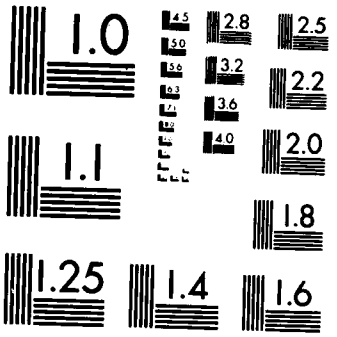
The Butterfly is a tightly coupled "shared-memory" architecture machine with at most 256 processors in the Motorola family (e.g. MC68000). Each processor sits on a card with a process node controller (PNC) and a megabyte of memory (which in some sense is local to this processor). Processors are interconnected in a "butterfly" fashion, thus enabling processors to gain quick access to memory located physically on a different card. There is a single clock in the whole machine but asynchronous multiple instructions may be executed by the different processors on multiple data.

One standard approach to programming the Butterfly is the so-called Uniform System [12]. The Uniform system views memory as one huge shared memory seen in the address space of all identical processors and seeks to scatter application data to reduce memory contention. Parallelism is exploited by generating tasks using a "generator" (similar to the **map** function in LISP). This is typically visualized as a procedure which has as its arguments a description of the data and a worker function to analyze the data; the generator results in activating processors with different subsets of the data. The form of this subset is generally limited in the current release (version 2.2.3) and typically is a row of a matrix.

Parallelism in the Butterfly suffers because of memory contention, switch contention and overhead involved in generating the tasks. Unfortunately, in spite of the existence of a scheduler, it is not possible to generate more active processes than processors in the Uniform system.

AD-A188 106 AN OVERVIEW OF VISION-BASED NAVIGATION FOR AUTONOMOUS 2/2
LAND VEHICLES 1986..(U) MARYLAND UNIV COLLEGE PARK
CENTER FOR AUTOMATION RESEARCH S CHANDRAN ET AL.
UNCLASSIFIED APR 87 CAR-TR-285 ETL-0479 DACA76-84-C-0004 F/G 12/9 NL





6.2 Butterfly Hough transform

The implementation of the Hough transform for detecting straight lines involves incrementing, for each edge pixel, the counts in all accumulator array entries that correspond to lines passing through that edge pixel. The time required to execute this algorithm is proportional to the size s of the grid, plus the number m of edge pixels times the number of votes v cast by each point i.e., O $(s+mv)$. The memory space is proportional to the size of the grid.

Once the Hough array is computed, those array elements which have high values must be identified. However, it is not sufficient to choose the few highest values because, depending upon the quantization, the same edge pixel will contribute to a "thick" curve, i.e., votes to entries that are adjacent. More importantly, because of the inherent non-linearity of the transformation, a "real" cell will be surrounded by "false" votes in its neighborhood. These problems are solved by searching for the local maximum in an $I \times I$ square neighborhood, perhaps after smoothing the Hough array.

In the simplest case, the steps involved in detecting large collinear sets of edge or feature points in a binary picture may be summarized as follows: Let $\theta_{\min}$ to $\theta_{\max}$ be the range of angles of interest and let *Accum* refer to the accumulator array. Then the simple algorithm is :

for each edge point (x,y)

for θ=θ_{min},θ_{max},$\Delta\theta$,

begin

$\rho = \lceil x \times \cos\theta + y \times \sin\theta \rceil$

Accum $[\rho,\theta]$= Accum $[\rho,\theta]$+1

end

Once the accumulator array is computed, we must next find the k highest local maxima. One can then construct another array which has as its entries all those elements which are the local maxima in all 3×3 neighborhoods. Next we could find the first k values (for instance by sorting the newly constructed array). In the sequential case, there is an obvious better algorithm which skips the intermediate step and directly finds the first k peaks. This is done using an array of k items (called final_list) which stores the angle, distance and Hough entry values.

On a systolic machine, such as the Warp, one may be interested in detecting the peaks even while the Hough array is being calculated, but on our machines there is no obvious advantage to this approach. We construct the Hough array and detect the peaks in a sequential manner but in both of the steps we would like to allocate processors optimally.

6.2.1 Computing the Hough array

Let θ_{min} to θ_{max} determine the relevant range of angles. One may think of computing the Hough array in terms of mapping a three-dimensional $x \times y \times \theta$

space into a one-dimensional ρ space and then incrementing the appropriate (ρ,θ) cells by 1.

At this stage, it is not obvious how the processors should be allocated. For instance, we can identify the following options:

- Distribute processing by x: Assign one row of the binary image to each processor. It then computes ρ over the entire range of angles. (One could conceivably use a column instead of a row.)
- Distribute processing by region: Assign a square shaped region of the input binary image to a processor. It then computes ρ over the entire range of angles.
- Distribute processing by θ: Assign one angle to each processor. It then computes ρ for the entire binary image.
- Distribute proc ssing by θ: As above, assign a processor to an angle. However, the processor analyzes only a part of the input image.

As an example, consider scaling the problem down to an input picture of size 1×2, an angle spread of 5° and an angular quantization of 1° (an assumption we will make throughout, for simplicity). Assuming that we have only two processors, we immediately see that it is preferable to distribute processing by θ. The only thing that we can hope for is that the input image is random. If we allocate one processor to each of the pixels, then one processor may end up doing no work at all since it may not be the case that both the pixels are edge pixels. On the other hand, distributing by θ implies that both processors are working at all

times and will, in general, lead to better processor allocation. This corresponds to the case of "Distribute processing by θ" above and will be the pivotal idea on which we shall base the allocation.

We present an allocation scheme for the general case (other schemes are also possible). Let $\Psi \equiv \theta_{\max} - \theta_{\min} + 1$ and let N be the number of rows in the input image. We will consider the following three possible cases:

$$P < \Psi \tag{1}$$

$$\Psi < P < \Psi N \tag{2}$$

$$\Psi N < P \tag{3}$$

In the first case, the ith processor computes the Hough array for a set of θs; specifically, $\theta_{\min}+i$, $\theta_{\min}+i+P$, . . . , $\theta_{\min}+i+\lceil \Psi (modulo P) \rceil$[1] over the entire binary image.

In the second case, the unit of processing given to a processor is a row of the input image and an angle. For each edge point in the row, the processor computes (or looks up in a table) the appropriate value of ρ. Each processor can be given either overlapping sets of rows or "scattered" sets (our implementation).

Finally, in the last case, the unit of processing is now an angle and part of the input row. Thus, for instance, if $P = 2\Psi N$, then processors will be working on an angle and half a row of the input image.

We omit the details of the complexity analysis and how our algorithm compares with any other. Details may be found in [13].

[1] A different partitioning where each processor analyzes contiguous angles is possible.

6.2.2 Implementation details

We remarked earlier that there are other data allocation schemes that may also achieve similar performance. Our scheme has the additional merit that on the Butterfly it is easy to implement. When the Uniform System generator produces tasks to be executed in parallel, it produces, conceptually, an *identification* number for each task and different processors execute tasks corresponding to different identification numbers. In our situation, when we allocated angles to each processor (corresponding to equation (1) in Section 6.2.1), these numbers correspond to the different angles since now the unit of processing is an angle. As a result, the P processors evaluate the first P tasks in parallel and then process the next P tasks. (Each processor thereby does not work on the task corresponding to contiguous angles.) In the next situation (see equation (2) in Section 6.2.1) we decided to allocate rows of the input image for a similar reason and thus each processor does not work on contiguous rows but rather on a scattered set of rows. It also helps that adjacent elements in a row of the matrix are also physically adjacent even though adjacent rows are scattered since *block* transfer of physically adjacent memory cells from global memory to the local memory of the processor is efficient on the Butterfly. Thus there is no contention when different processors are accessing the input image in parallel.

Once the individual Hough array versions are computed by the different processors, we need to merge them into one "correct" Hough array. Communication between processors on the shared memory Butterfly is achieved by directly accessing global memory. Thus, our implementation defined the "final" Hough

array in global memory. The individual versions are never explicitly built but instead the copy in global memory is updated. It is well known that updating in global memory requires that the information be added atomically; so we use the "locking" mechanisms available in Chrysalis to ensure integrity. In particular, we use the *Fetch-and-Add* instruction to trivialize the atomicity problem. Also, there is no need to allocate memory in the global address space for the individual versions. One also manipulates the program variables to take advantage of the local and global memory aspects of the shared memory and reduce the contentions. Since programming on the Butterfly is done in C, we often need to follow a sequence of pointers in order to access the "locked" data structure. A useful strategy in such cases is to make local copies of the data thus restricting contention only at the final destination memory word. In fact, for the Hough transform done on a window of an image for the Autonomous Land Vehicle, we observed linear speedup for a 32×32 input binary image with approximately twenty-five percent edge pixels when the angle spread was about twenty degrees.

The same idea may be used even in the processor allocation step. In addition, accesses to local memory cost less than accesses to global memory; so a "block" of data may be transferred to local memory and then used in computations. Again, when processors are allocated parts of the input matrix, allocation is done *alternatingly from the top and the bottom.* This results in less contention while accessing common data, for instance, overlapping rows.

6.2.3 Peak detection

More interesting theoretical ideas are involved in searching for the k highest peaks once the accumulator array is available.

At first, one is inclined to sort the Hough bins using the accumulator value as the key. (Once sorting is done, it is trivial to pick the k highest peaks.) Using N processors, we cannot sort in less than $O(\log N)$ time [14]. As far as implementations go, although the above is fast, it is not practical because the constant hidden in front of the $O(\log N)$ description is very large. The value of k in most applications is low and the following algorithm is an efficient alternative (though asymptotically not faster than sorting).

Processor allocation is done by splitting (but not partitioning) the Hough array into P strips. When successive strips share a common row, then each pixel and its 3×3 neighborhood is contained in some single strip (except for the border pixels). Then each processor using a fast sorting algorithm finds the first k elements in its data. We then combine the individual k peaks in a tree-like fashion to produce the final desired k peaks.

This algorithm may be easily implemented on the Butterfly. It is easy to observe that this algorithm is

$$O(N/P\log(N/P)+k\log P),$$

the first term arising from the sorting part and the second from the merging. Asymptotically, this is not a good algorithm but for "reasonable" values of k and N, it is satisfactory.

6.3 Parallel search

In this section we will confine our attention to blind, parallel tree search, and not consider the complications encountered when the search space is not a tree, or when heuristics are available to direct the search. A common approach to such parallel search problems is to maintain a global OPEN list in which unexpanded, nonterminal nodes in the search tree are stored. If a processor is available, then a task is generated by deleting a node from this OPEN list. The task determines whether or not the node is a goal node and if not, expands the node and places its successors onto the OPEN list. This approach, while simple, suffers from the following shortcomings:

1) Since there is only one source for task generation, processor utilization may be low. This problem is especially severe if the amount of computation that a task performs in node expansion is small.

2) The OPEN list is a very competitive data structure. This problem is more important as the number of processors increases.

To overcome these problems, we observed that since the search is not directed towards any specific subspace of the search space by heuristics, there is no need to maintain a global OPEN list. Each subtree can initiate and control its search independently of other subtrees. So, each task has a local OPEN list which initially contains the root node of the subtree that that task is to search for goal nodes. When a node is expanded, each descendent is either placed on this local OPEN list or is used to initiate a new task. This decision is based on the availability of free processors at the time of node expansion. If there are free

processors, then a new task is spawned; otherwise, the generated nodes are placed onto the local OPEN list. Using this approach, processors are fully utilized most of the time even if there are a large number of processors. This approach also spawns considerably fewer tasks than the simple approach which generates as many tasks as nodes. The local OPEN list algorithm only generates a task when there is an available processor. This is especially advantageous when the task generation overhead is high, as it is in the early releases of BLISP (approximately 20–40 ms per task generation).

Although there is no OPEN contention problem with local OPEN lists, there is some contention for accessing and modifying processor status conditions. This problem can be reduced by checking processor status less frequently (i.e., not *after the generation of each descendent of a node being expanded, but, say, after* the generation of a subtree of depth k rooted at that node). However, this can result in lower overall processor utilization.

We have applied this parallel search algorithm to a simplified version of the search algorithm presented by Grimson and Lozano-Perez[15] for object recognition. The simplification was originally used by Harris and Flynn[16] for developing Connection Machine algorithms for parallel search. The problem can be briefly described as follows: An object is modeled by a "fuzzy" set of points; for each pair of points we know a range of distances that can separate those points. Given any set of data points from an image, the goal of the search procedure is to find all mappings between the data points and model points that satisfy all of the pairwise distance constraints in the model. Table 6.1 shows the relationship

Table 6.1 Parallel search processing times

global OPEN algorithm		
processors	time in sec.	effective processors
1	349.2	1
5	73.0	4.8
10	38.2	9.1
15	26.4	13.2
20	24.4	14.3
25	24.0	14.5
30	24.6	14.1
35	23.4	15.0
40	23.9	14.6
45	23.6	14.8
50	23.6	14.8

local OPEN algorithm		
processors	time in sec.	effective processors
1	309.8	1
5	68.8	4.5
10	33.9	9.1
15	26.3	11.8
20	22.0	14.0
25	19.3	16.1
30	15.4	20.1
35	13.8	22.4
40	14.1	22.0
45	12.4	25.0
50	12.1	25.6

between number of available processors and computing time for a problem involving a model containing four points and a data set containing seven points for both the global OPEN and the local OPEN algorithm. Overall address space limitations of Butterfly LISP did not allow us to utilize more than 50 processors in these experiments. The last column in the tables, labeled "effective processors," is simply the ratio of the time it took to solve the problem on a single processor to the time it took to solve the problem for "processors" processors. While the absolute speeds of the program are very poor (due mostly to the rather large overhead of task generation in BLISP), the local OPEN program does exhibit reasonable scaling properties with respect to the number of processors.

In Huang and Davis[17] we present more results for this search procedure, as well as a discussion of parallel heuristic search algorithms for the Butterfly.

Figure 7.1 Curving Road

Figure 7.2 Detected boundaries

7. Experimental Results

The vision system described in Section 3 has been used to drive the robot arm over all portions of our terrain board, including the turns and intersections. In Section 7.1 we illustrate its behavior using two examples—one in which the camera is driven around a curve (illustrating the sensor control to maintain maximum visibility of the road), and the second in which the vehicle turns at an intersection.

Additionally, we have conducted two sets of experiments with parts of this vision system on the Martin Marietta vehicle. These experiments are described in Section 7.2.

7.1 Simulator experiments

Figures 7.1 shows an image of a curving road as seen from the perspective of the camera on the robot arm. The feedforward image processing algorithm is used to identify the boundaries of the road, and in Figure 7.2 we show the superposition of those detected boundaries on the image in Figure 7.1. A three-dimensional model of the turning road is built using the appropriate inverse perspective algorithm, and a motion of the vehicle through the beginning of the turn is computed. Based on the predicted location of the vehicle after this motion, a new relative sensor heading is computed by the sensor control module. In Figure 7.3 we show the image of the road taken by the camera from the next viewing position, with the new sensor heading. For comparison, in Figure 7.4 we show the image that would have been obtained had the vehicle been moved to this new

Figure 7.3 Road image from next vantage point with sensor control

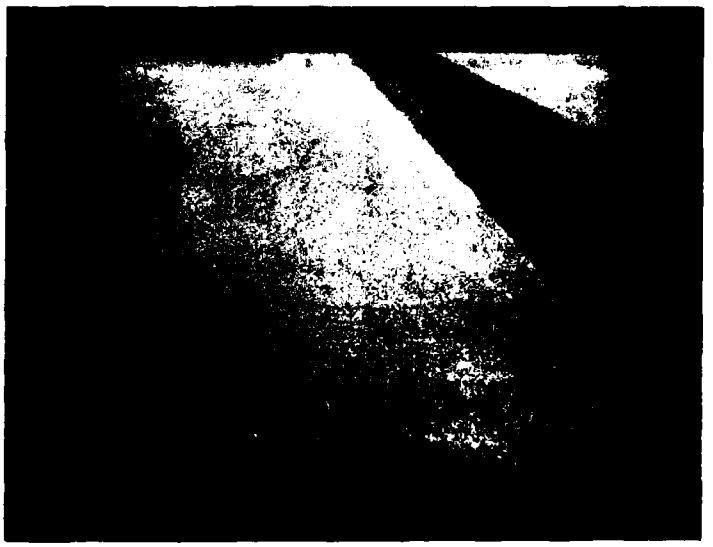


Figure 7.4 Same as Figure 7.3, but without sensor control

viewing position, but the sensor heading remained unchanged.

The second example illustrates the negotiation of an intersection by the vision system. In Figure 7.5 we show the image obtained by the camera as it approaches an intersection. Based on an estimate of the vehicle's position, and information in the map indicating the location and structure of the intersection, the vision executive is able to place a window in the image that contains the intersection. Although this window is itself never analyzed by the image processing module, its location is used to place other windows predicted to contain the boundaries of the intersecting road. These windows are illustrated in Figure 7.6a. The complete results of the feed-forward image processing are shown in Figure 7.6b. As above, a three dimensional model of the road structure is developed, and the vehicle moves a short distance towards the intersection. Figure 7.7 shows the image obtained from the next vantage point, as well as the superposition of the three dimensional road model computed from the previous frame. This model is used to generate predictions of the locations of both the road on which the vehicle is traveling as well as the intersecting road.

7.2 Experiments at Martin Marietta

Two sets of experiments were conducted using the Martin Marietta vehicle, one in November of 1985 and the second during the summer of 1986. The first set of experiments involved a very limited portion of the feedforward vision system. The only components included were a part of the vision executive (for placing all but the first two windows for the feedforward processing) and the image

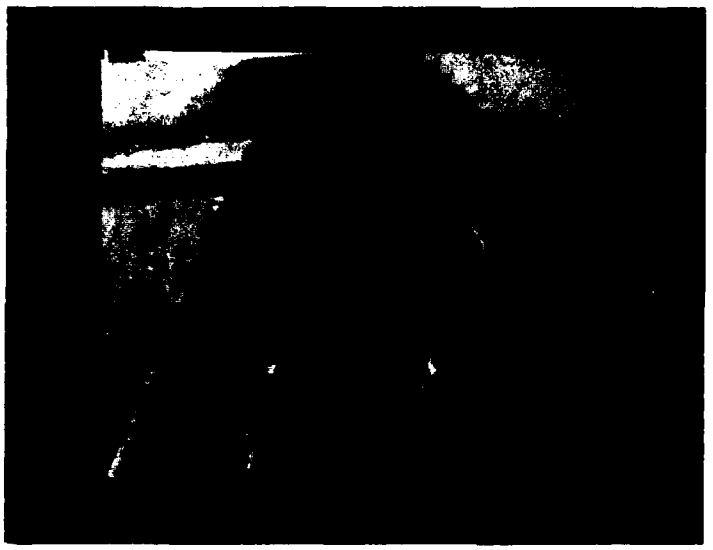


Figure 7.5 Intersection

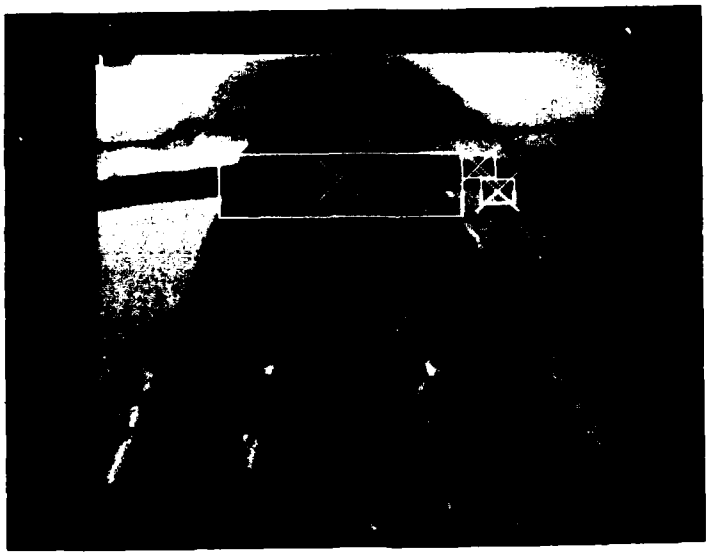


Figure 7.6a Forbidden region and prediction windows

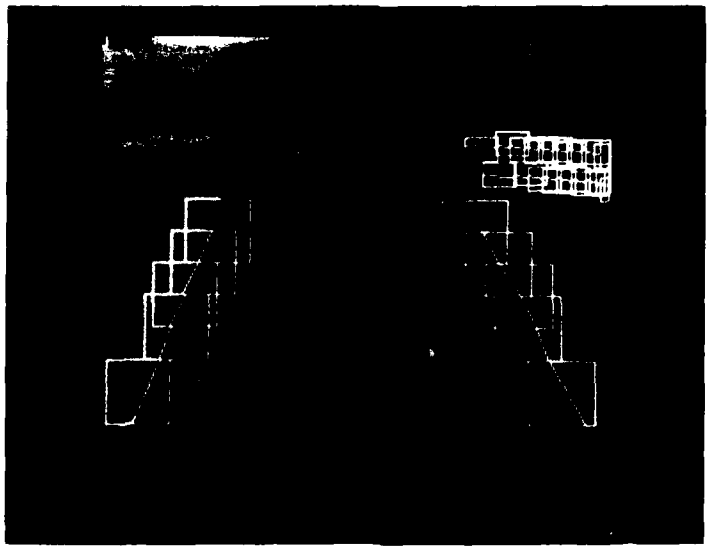


Figure 7.6b Extracted road boundaries

Figure 7.7 Image from next vantage point

processing for linear feature extraction. Since the predictor was not included, the system placed relatively large windows near the bottom of the image to ensure that both road boundaries were contained in the initial windows. These windows were 64×32 windows. As a result of using such large windows, processing took on the order of 10–11 seconds per frame. The vehicle was driven over a few hundred yards of the test track at a speed of approximately three kilometers per hour. The vehicle eventually ran off of the road due to accumulated errors in the initial window placement.

During August of 1986 a much expanded version of the feedforward vision system was tested in Denver. The set of image processing algorithms was extended to include the region based algorithms described in Section 3, and the algorithms were applied to significantly reduced resolution versions of the full frame TV color image. In these experiments, segmentations were obtained for road images subsampled to 64×64 pixels, and the left and the right road edges were selected simultaneously based on *a priori* knowledge of geometric road properties—here, the road width. The algorithm used knowledge about the predicted location of the road to position a window at the lower part of the road, which was assumed to contain only road pixels. Statistics of the pixels in this window were computed to determine the parameters of the pixel classification algorithm that made a road/non-road decision on a pixel by pixel basis. The entire image was then segmented using this classification algorithm. Next, a window predicted to contain segments of both the left and right road boundaries was computed using the prediction module. Line segments were fit to

the border by simultaneously determining for the window the road/non-road border points along the top, middle and bottom rows of the window, as described in Section 3. Subsequent windows were placed to maximize the length of the road boundary. Processing took approximately two seconds per frame, including about .75 seconds for processing the initial window. The vehicle ran for several hundred yards along the test track at a speed of 5 kilometers per hour.

References

1. A. Waxman, J. LeMoigne, L. Davis, E. Liang and T. Siddalingaiah, "A visual navigation system for autonomous land vehicles," to appear in *IEEE Transactions on Robotics and Automation, 1987.*

2. H. T. Kung and J. Webb, "Mapping image processing operations onto a systolic machine," *Distributed Computing,* 1, 246–257, 1987.

3. "The Butterfly parallel processor," Bolt, Beranek and Newman, 1986.

4. D. Hillis, *The Connection Machine,* M.I.T. Press, Cambridge MA, 1985.

5. L. Davis, T. Kushner, J. LeMoigne, A. Waxman, "Road boundary detection for autonomous vehicle navigation," *Optical Engineering,* 25, 409–414, 1986.

6. D. Dementhon, "Inverse perspective of a road from a single image," University of Maryland Center for Automation Research TR-210, July 1986.

7. S. Kambhampati and L. Davis, "Multiresolution path planning for mobile robots," *IEEE Transactions on Robotics and Automation,* 2, 135–145, 1986.

8. H. Samet, "The quadtree and related hierarchical data structures," University of Maryland Center for Automation Research TR-99, November 1984.

9. S. Puri and L. Davis, "Two dimensional path planning with obstacles and shadows," University of Maryland Center for Automation Research TR-255, January 1987.

10. S. Puri and L. Davis, "Road detection and following on the Butterfly," University of Maryland Center for Automation Research TR, in preparation.

11. R. Halstead, "Implementation of MultiLisp: Lisp on a Multiprocessor", 1984 ACM Symposium on LISP and Functional Programming, Austin Texas, 9–17, 1984.

12. W. Crowther, "Using the Uniform System to program the Butterfly," Bolt, Beranek and Newman, 1986.

13. S. Chandran and L. Davis "The Hough transform on the Butterfly and the NCUBE," University of Maryland Center for Automation Research TR-226, September 1986.

14. M. Ajtai, J. Komlos and E. Szemeredi, "An n log$n$ sorting network,"' *Combinatorics,* 3, 1981.

15. W. Grimson and T. Lozano-Perez, "Model-based recognition and localization from sparse range or tactile data," M.I.T. Artificial Intelligence Memo 738, August 1983.

16. J. Harris and A. Flynn, "Object recognition using the Connection Machine's router," Computer Vision and Pattern Recognition Conference, Miami Florida, 134–139, 1986.

17. S. Huang and L. Davis, "Parallel search algorithms on the Butterfly parallel processor," University of Maryland Center for Automation Research TR, in preparation.

END

FILMED

MARCH, 1988

DTIC